UNIVERSITÀ DI PISA

Facoltà di Ingegneria

Corso di Laurea Specialistica in Ingegneria Elettronica

Nuovo Ordinamento

Tesi di Laurea

Rete di sensori wireless per la misura dei bordi liberi su imbarcazioni da diporto

Relatori:

Prof. Roberto Saletti

Ing. Federico Baronti

Candidato:

Gabriele Fantechi

Anno Accademico 2008 - 2009

Contenuti

1. Introduzione

1.1. Introduzione

La misura dei bordi liberi, ossia la distanza verticale tra il ponte di coperta e la linea di galleggiamento di un'imbarcazione, è un'importante verifica che viene effettuata su tutte le imbarcazioni, siano esse a vela o a motore, una volta varate.

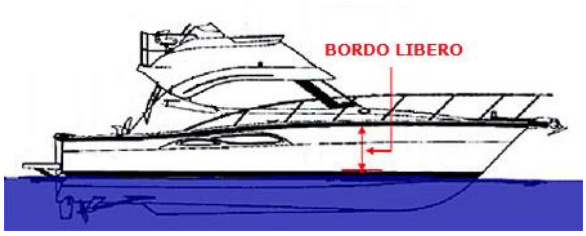


Figura 1. Bordo libero

Dalla conoscenza della misura del bordo libero in diversi punti dello scafo è possibile avere informazioni sull'assetto statico che l'imbarcazione assume. Questi elementi sono essenziali per poter certificare la rispondenza alle specifiche che i progettisti hanno definito sulla carta. In caso di discordanza con quanto previsto, si possono ricavare indicazioni utili per apportare le dovute correzioni, ad esempio dislocando diversamente i pesi a bordo, oppure modificando la forma stessa dell'opera viva (superficie dello scafo immersa).

Queste misure acquistano interesse anche maggiore se effettuate su una barca a vela. L'assetto che lo scafo assume influenza fortemente (più di quanto accade su imbarcazioni a motore) la stabilità dell'imbarcazione in navigazione, nonché le sue prestazioni (velocità, capacità di stringere il vento). Le informazioni che si possono trarre dalla misura possono essere utilizzate per effettuare correzioni e modifiche, tenendo conto che in questo caso si hanno molti più gradi di libertà: oltre allo spostamento dei pesi a bordo, è possibile agire sulle regolazioni delle manovre dormienti per cambiare l'assetto dell'albero, o ad esempio, in fase di progetto, modificare la forma e/o la posizione della zavorra sotto la chiglia. Inoltre possono essere estratte indicazioni utili per il taglio e il confezionamento delle vele.

Le misurazioni oggi vengono effettuate in maniera indubbiamente rudimentale: l'operatore si serve di un asta metrica che viene posizionata verticalmente nel punto da misurare, quindi

viene effettuata la lettura del valore ad occhio nudo, cercando di mediare le oscillazioni causate dai movimenti dell'imbarcazione e della superficie del acqua. Quindi la misura viene ripetuta in posizioni diverse: solitamente su un'imbarcazione a motore vengono effettuate tre misure, una a prua, e due a poppa, a dritta e a sinistra[1].

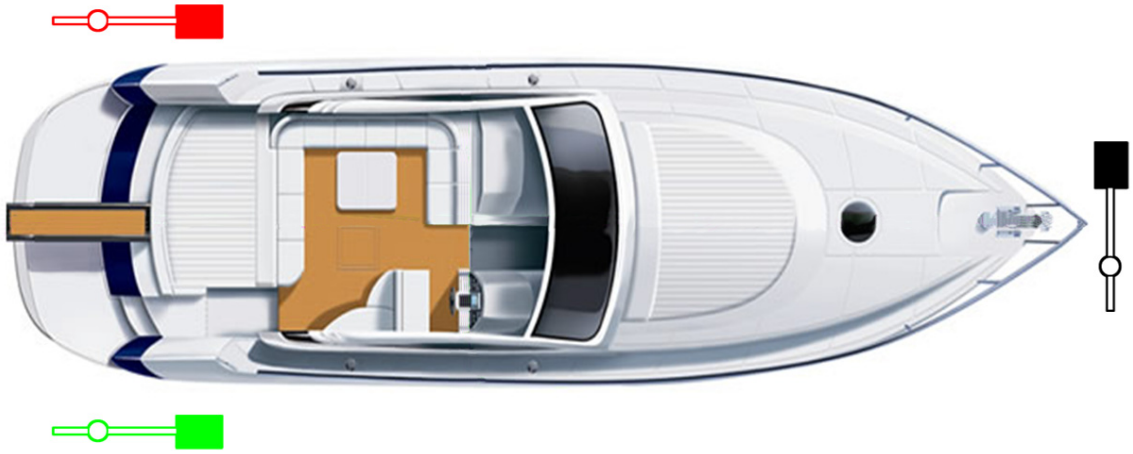


Figura 2. Posizione delle unità di misura a bordo dell'imbarcazione

Questo metodo è affetto da numerose sorgenti di errore, a cominciare dalla lettura effettuata ad occhio nudo, ma anche dal fatto che le varie misure vengono effettuate in istanti diversi. Inoltre è praticamente indispensabile la presenza a bordo di uno o due operatori: la variazione di assetto causata dalla loro presenza può essere trascurabile qualora si misuri un luxury yacth di 60 piedi (non verrebbe comunque apprezzata a causa della scarsa risoluzione dello strumento). Se però i rilevamenti sono effettuati a bordo di una barca a vela da competizione di poche decine di piedi, il peso di una persona sulla prua può modificare sensibilmente la valutazione. Tutte queste fonti di indeterminazione fanno si che i risultati delle misure siano affetti da un errore non inferiore al centimetro.

In questo lavoro di tesi si vuole indagare la possibilità di **effettuare questa misura in maniera automatica**, in modo da semplificare le operazioni da effettuare e, soprattutto, **ottenere una migliore accuratezza** sulla misura, dell'ordine del **millimetro**. Si è pensato di utilizzare una rete senza fili per connettere diversi sensori opportunamente posizionati a

[1] Altre procedure prevedono la doppia misura anche a prua.

bordo e controllati da un PC. Ognuno di essi effettua la misura nel punto in cui è posizionato e i risultati sono disponibili sull'unità centrale. Con un sistema del genere non è richiesta la presenza di operatori a bordo in quanto le operazioni di misura possono essere controllate a distanza, ad esempio dalla banchina. Questi sensori devono essere in grado di rilevare l'altezza di bordo libero a prua e a poppa, sia a destra che a sinistra. Per validare la misura ed eventualmente depurarla dai movimenti di rollio e beccheggio dell'imbarcazione è utile poterne controllare anche l'inclinazione sui due assi. Per questo motivo nella rete di sensori viene inserito anche un inclinometro.

Per costruire il sistema è necessario individuare una famiglia di sensori in grado di misurare il livello di un liquido. Questi devono garantire affidabilità e ripetibilità della misura, e devono essere robusti e resistenti ad un ambiente aggressivo come quello salmastro. Quindi è necessario realizzare una scheda per alimentare i sensori mediante una batteria, e per interfacciarli con il PC. Anche per l'inclinometro devono essere seguiti gli stessi criteri.

Dovrà essere curato lo sviluppo di software in grado di registrare i dati delle misure e presentarli all'utente.

Obiettivo ultimo è quindi la realizzazione di un sistema versatile, di facile installazione ed utilizzo, controllabile con un'interfaccia facilmente comprensibile con cui l'utente possa effettuare contemporaneamente tre misure di bordo libero, con grande precisione.

Nei successivi capitoli viene descritta più nel dettaglio l'intera rete (capitolo 2), quindi sono introdotti i vari dispositivi che compongono la rete, ossia inclinometro (capitolo 3) e misuratori di livello (capitolo 4). É proposta una soluzione software con la quale controllare la rete e infine sono descritti i risultati ottenuti durante una prova sul campo, a bordo di un'imbarcazione (capitolo 5).

1.2. Elettronica per applicazioni nautiche

L'elettronica applicata al settore della nautica è, al giorno d'oggi, in continua crescita ed evoluzione. Negli ultimi anni le imbarcazioni hanno iniziato ad ospitare a bordo ogni sorta di dispositivo elettronico, dagli strumenti di navigazione, a quelli di comunicazione, ma anche, sempre più spesso, sistemi di entertainment. Sebbene qualcosa di analogo sia già accaduto

in ambito automobilistico, i sistemi che devono essere alloggiati su un imbarcazione presentano differenze sostanziali. Innanzitutto tali apparati si troveranno esposti ad ambienti ostili, sia che vengano installati in locali chiusi, sia che siano collocati all'aria aperta: in prossimità di acqua salata infatti i processi di ossidazione e deterioramento delle parti metalliche sono notevolmente accelerati. Devono essere affrontati vari problemi nella realizzazione degli isolamenti galvanici tra gli apparati, tenendo conto delle difficoltà nel realizzare impianti di messa a terra. Può essere necessario tenere in considerazione anche l'aspetto del risparmio energetico e della generazione di energia da fonti rinnovabili, soprattutto nei sistemi dedicati alle imbarcazioni a vela. Particolare attenzione deve essere prestata quando l'elettronica interagisce strettamente con parti meccaniche, tenendo conto della continua variabilità dell'assetto dell'imbarcazione.

Nella quasi totalità delle imbarcazioni da diporto ad oggi attive, siano esse yacht di lusso, barche a vela da competizione o da crociera, tutti i sistemi elettronici sono installati a bordo come unità a sé stanti. Fatta eccezione per il bus di comunicazione che spesso collega i sistemi di navigazione (pilota automatico, stazione del vento, ecoscandaglio, solcometro, GPS, etc), non esiste un sistema centralizzato che tenga sotto controllo tutti gli altri apparati (motori, illuminazione, comunicazioni radio, etc..) come invece ormai accade anche nelle autovetture di fascia media. Questa mancanza di integrazione è dovuta a diversi fattori: sicuramente l'espansione del mercato dei sistemi elettronici per la nautica da diporto è iniziata con almeno dieci anni di ritardo rispetto al settore automobilistico; a questo si aggiunge la mancanza di standardizzazione dei vari sistemi e la disuniformità dei protocolli di comunicazione usati. I produttori di elettronica per la nautica cercano piuttosto di utilizzare standard customizzati e brevettati, magari uguali su tutta una serie di prodotti, per indirizzare la scelta dei sistemi da installare verso la serie stessa.

Questa mancanza di uniformità nasce insieme alla costruzione dell'imbarcazione stessa all'interno del cantiere. Le fasi della costruzione sono ancora oggi svolte quasi completamente per via artigianale, da operai esperti, saldatori, carpentieri. Ogni esemplare che viene varato non sarà mai uguale agli altri ma vi saranno scarti e tolleranze dovute proprio alla tipologia di lavorazione. Se a questo si aggiunge la possibilità di

personalizzazione pressoché infinita che i vari marchi offrono ai propri armatori (soprattutto sulle imbarcazioni di lusso), risulta evidente quando le imbarcazioni che navigano oggi siano diverse tra di loro.

Per tutti questi motivi ogni imbarcazione può e deve essere considerata come una singola unità e non il prodotto di una serie. Inevitabilmente anche le procedure di verifica e collaudo dovranno tenere conto di tutte le caratteristiche peculiari dell'imbarcazione.

Il Dipartimento di Ingegneria dell'Informazione dell'Università di Pisa si è inserito negli ultimi anni in questo settore, grazie anche a contratti di ricerca stipulati con noti marchi della cantieristica navale italiana, quale ad esempio Ferretti S.p.A., progettando e sviluppando sistemi in grado di gestire questa disuniformità dei sistemi elettronici e delle imbarcazioni stesse. In particolare è stato messo a punto un sistema capace di controllare e guidare il tecnico collaudatore durante una prova in mare, effettuando continuamente misure e acquisizioni da sensori presenti a bordo ed evidenziando peculiarità e anomalie della prova. Caratteristica fondamentale di tale sistema è la possibilità di essere utilizzato indipendentemente dalla tipologia di strumenti e di protocolli di comunicazione presenti a bordo: al variare delle motorizzazioni, delle configurazioni di assetto, dei sistemi di navigazione presenti, il sistema viene facilmente adattato al caso. Questo sistema, nato con il nome di Yacht Supervisor [1] e successivamente denominato BlackBox[2] sarà analizzato nel paragrafo successivo.

1.3. Il progetto BlackBox per Ferretti

BlackBox è un sistema elettronico in grado di monitorare tutti i dati provenienti dai sensori e i sottosistemi di interesse presenti sull'imbarcazione e rappresentarli in un'unica interfaccia grafica di facile utilizzo e interazione da parte dell'operatore. L'acquisizione delle grandezze viene effettuata in tempo reale, inoltre è prevista l'archiviazione dei dati su file in opportuni formati, per permettere successivi recuperi ed elaborazioni.

In questo modo si vuole ottenere una valutazione **oggettiva** della prova di collaudo che lo yacht deve superare.

[2] Presentato alla stampa nel Luglio 2009 col nome di F.A.I.R. (Ferretti Advanced Integral Recorder).

Il sistema può essere suddiviso in due sezioni: una parte hardware e una software. La prima effettua l'acquisizione dei segnali provenienti dai sensori di bordo e la conversione dei protocolli di trasmissione dati dei vari sottosistemi nel formato standard adottato per il progetto. Questa parte comprende l'utilizzo di opportuni circuiti e supporti di trasmissione dati, nonché dello sviluppo di circuiti custom per la misura di dati non disponibili a bordo. La seconda è invece finalizzata all'acquisizione dei dati mediante i canali di trasmissione, decodifica, elaborazione, visualizzazione e archiviazione. Questa fase viene eseguita mediante un'applicazione software sviluppata con il pacchetto *LabVIEW* di National Instruments. La struttura del sistema è riportata in Figura 2.

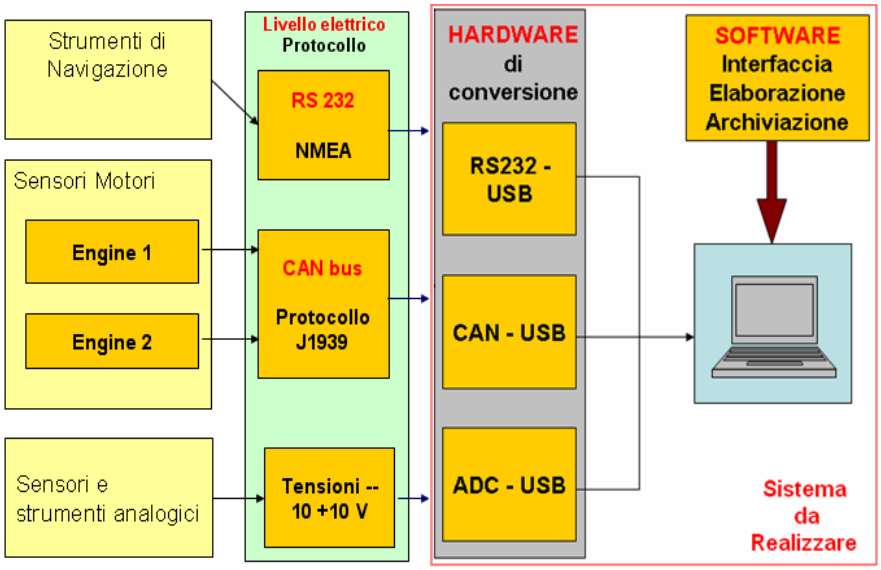


Figura 3. Architettura del sistema Pershing Yacht Supervisor, precursore del sistema Black Box

L'hardware del Pershing Yacht Supervisor comprende le interfacce di acquisizione dati dai vari sistemi di bordo: la scelta è stata quella di trasformare i livelli fisici sui quali sono trasportate le informazioni, indipendentemente dalla loro sorgente, in dati compatibili con lo standard Universal Serial Bus (USB). Per l'acquisizione dei dati dai motori e dalle sorgenti analogici si ricorre a sistemi di interfaccia già esistenti in commercio. I segnali sono poi convogliati in un hub USB verso il PC su cui è eseguita l'applicazione software.

Dati	*Tipo dati*	*Prelievo*	*Interfaccia acquisizione*
Dati motore dx	Digitale (stringa 16 b)	CAN Bus (J1939)	Lawicel CANUSB

Dati motore sx	Digitale (stringa 16 b)	CAN Bus (J1939)	Lawicel CANUSB
Coordinate geografiche, orologio	digitale (stringa)	NMEA 0183	RS-232 to USB
COG, SOG	digitale (stringa)	NMEA 0183	RS-232 to USB
Angolo timoneria (RSA)	digitale (stringa)	SeaTalk to NMEA 0183	RS-232 to USB
Inclinazione X,Y dello scafo	digitale	Bluetooth	Bluetooth
Sensori potenziometrici Flap sx,dx	analogico (0.6-5.9 V)	Indicatore Flap	NI DAQ USB-6008
Sensori potenziometrici Trim sx,dx	analogico (1.5-6.1 V)	Indicatore Flap	NI DAQ USB-6008

Tabella 1. Elenco dei segnali ricevuti dalle sorgenti da monitorare e interfacce hardware

Per misurare le inclinazioni dell'imbarcazione durante la prova, il Dipartimento di Ingegneria dell'Informazione dell'Università di Pisa ha realizzato un prototipo di inclinometro biassiale senza fili, basato su sensore di accelerazione con tecnologia, alimentato a batteria (vedi capitolo 3); per i dati da navigatore GPS e Autopilot è stato sviluppato un circuito custom che effettua una conversione RS-232/RS-422→USB[3].

La sezione software è costituita da un applicativo sviluppato con il linguaggio di programmazione *LabVIEW* e schematizzato in Figura 4. I dati, acquisiti dalle varie sorgenti in tempo reale mediante le interfacce hardware, sono soggetti a una decodifica e interpretazione dei protocolli, finalizzata a una visualizzazione su interfaccia di presentazione grafica e archiviazione su opportuni file con frequenza di log di 1 Hz.

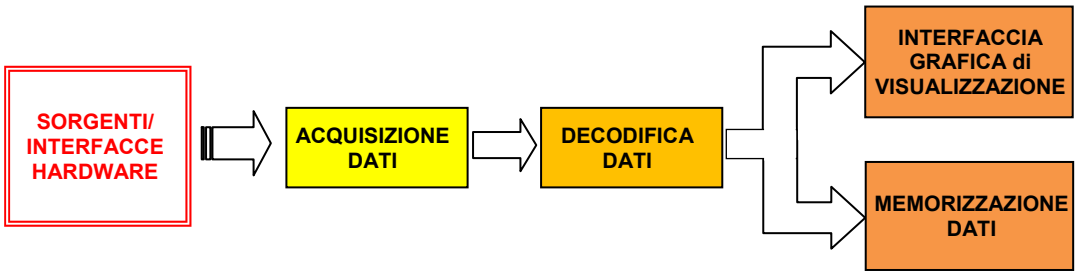


Figura 4. Funzioni svolte dalla sezione software del Pershing Yacht Supervisor

L'applicativo sviluppato garantisce all'utente una completa configurabilità della prova con controlli di attivazione della registrazione dati e selezione delle periferiche da acquisire.

[3] Nelle ultime version di BlackBox si utilizza un PC dotato di porta seriale RS-232.

L'applicazione può essere eseguita in diverse modalità (*Test*, destinata all'assistenza del collaudatore durante la prova a mare, *Failure Analisys*, un'interfaccia più completa che da accesso a tutti i dati acquisiti a bordo). L'aspetto grafico di una finestra dell'interfaccia è riportato in Figura 5.

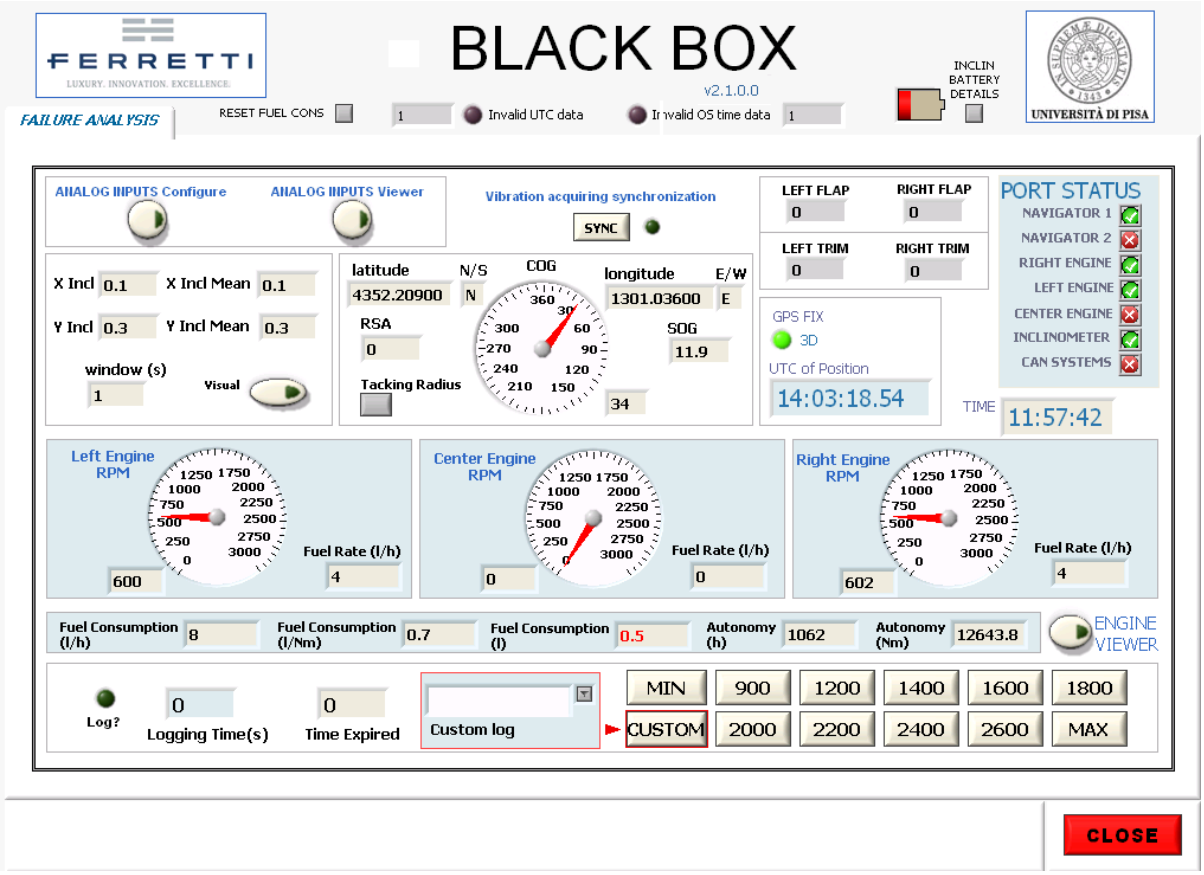


Figura 5. : Interfaccia grafica dell'applicativo Black Box in modalità *Failure Analysis*

In questo lavoro di tesi è stata messa a punto un'interfaccia di post-processing dei dati GPS che è stata integrata nell'ambiente BlackBox. Questa ha la funzione di calcolare il raggio delle virate dell'imbarcazione in tempo reale, durante la prova in mare. Per il calcolo si sfruttano i dati ricevuti dal ricevitore GPS contenuti nella stringa VTG[4]: velocità e angolo di prua.

Un esempio di stringa VTG è riportato di seguito. [2]

```
$GPVTG,054.7,T,034.4,M,005.5,N,010.2,K*48
```

[4] Vector Track over the Ground.

```
VTG          Identifier
054.7,T      True track made good (degrees), COG[5]
034.4,M      Magnetic track made good
005.5,N      Ground speed, knots, SOG[6]
010.2,K      Ground speed, Kilometers per hour
*48          Checksum
```

Dalle legge oraria del moto circolare uniforme: $v = r \cdot \omega$, ovvero $v = r \cdot \frac{d\theta}{dt}$. Invertendo questa relazione è possibile calcolare il raggio:

$$r = \frac{v \cdot dt}{d\theta}$$

Approssimando quindi la virata dell'imbarcazione come composta da piccoli tratti percorsi di moto circolare uniforme, quindi con velocità tangenziale e angolare costante, è possibile risalire al raggio semplicemente conoscendo la velocità tangenziale (v) , la variazione delle prua tra due istanti ($d\theta$) e durata dell'intervallo temporale tra i due istanti (dt).

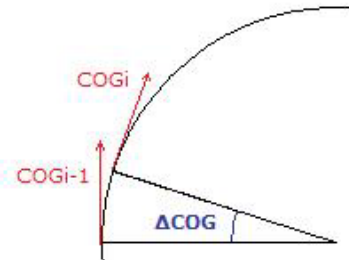


Figura 6. Determinazione della variazione della prua

Per rendere l'algoritmo insensibile alle fluttuazioni dei dati istantanei, i dati di COG e SOG sono mediati su una finestra mobile la cui ampiezza è determinata dalla relazione *n°campioni* x 0,2s. Il fattore *n°campioni* è impostato mediante il controllo *Windows Length*; il parametro 0,2 corrisponde alla distanza temporale tra un campione e l'altro in secondi[7]. Si ottengono quindi i valori mediati $\overline{COG}$ e $\overline{SOG}$, da cui

$$\Delta COG = \overline{COG}_{i+1} - \overline{COG}_i$$

[5] Course Over Ground
[6] Speed Over Ground
[7] Il ricevitore GPS utilizzato fornisce dati alla frequenza di 5 Hz: si avrà quindi un nuovo dato ogni 0,2 secondi.

I due valori $\overline{COG}_{i+1}$ e $\overline{COG}_i$ sono i *COG* mediati su due finestre temporali consecutive della stessa ampiezza, il cui istante iniziale differisce di un numero di campioni impostato mediante il parametro *Windows Distance*.

Il raggio ottenuto viene ulteriormente mediato su una finestra mobile la cui ampiezza temporale è determinata dalla relazione "*Points of AVG* x 0,2".

Agendo sui controlli *Windows Length*, *Windows Distance* e *Points of AVG*" si modellano i parametri per il calcolo del raggio di virata secondo le esigenze della prova a mare in corso e della tipologia di imbarcazione che si sta valutando.

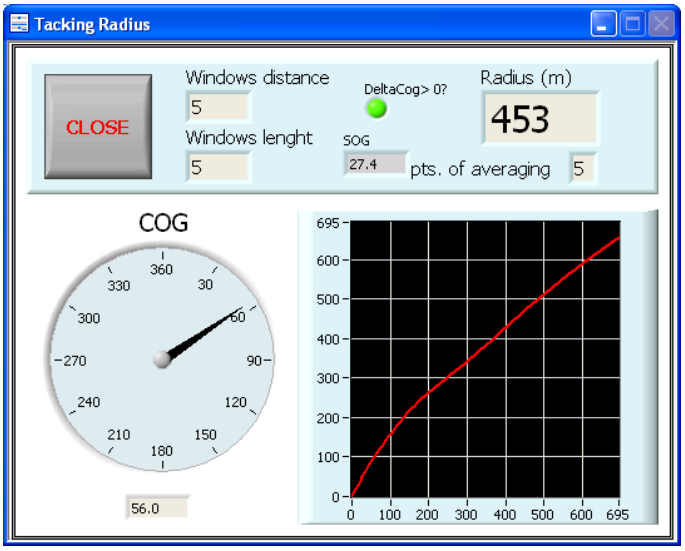


Figura 7. Visualizzazione del raggio di virata

Alla misura calcolata del raggio di virata si associa anche l'informazione sulla direzione della virata, codificata nel segno del valore presentato:

- raggio positivo: per rotazioni in senso orario, $\Delta COG > 0$
- raggio negativo: per rotazioni in senso antiorario, $\Delta COG < 0$.

E' stato inoltre introdotto un limite sull'ampiezza massima possibile del raggio di virata: valori del raggio superiori a 1000 m sono rappresentati con l'etichetta "Inf" (per traiettorie rettilinee infatti il raggio di virata tende all'infinito).

2. Sistema di misura dei bordi liberi

2.1. Descrizione e specifiche del sistema

In questo lavoro è stato sviluppato un sistema di misura dei bordi liberi, in grado di misurare contemporaneamente i bordi liberi a prua e a poppa dell'imbarcazione, con una precisione di 1 mm.

Il sistema è composto da una unità centrale e tre unità di misura remote: i collegamenti possono realizzarsi mediante rete cablata o rete wireless. All'unità centrale può anche essere connesso un inclinometro, per registrare i movimenti dell'imbarcazione durante le operazioni di misura.

Figura 8. I dispositivi della rete di misura di bordi liberi.

La versione wireless del sistema, realizzata in questo lavoro di tesi, permette di effettuare la misura senza la presenza dell'operatore a bordo: questi può infatti controllare la misura dalla banchina, tramite computer portatile o dispositivo palmare. Nella versione cablata questa configurazione è ottenibile con maggiori difficoltà operative, dovendo esser portata in banchina una connessione fisica alla rete di misura.

Per poter interfacciare la rete di misura con un notebook o un dispositivo palmare si è scelto di realizzare la rete wireless basandosi sulla connettività Bluetooth, oggi disponibile su praticamente tutti i dispositivi portatili. Tale protocollo di comunicazione è descritto brevemente nel paragrafo 2.3.

Per effettuare la misura è necessario posizionare correttamente le tre unità di misura. Volendo ottenere una buona affidabilità sulla misura si ricorre ad una misura di livello meccanica: i sensori misureranno la posizione di un cursore galleggiante, libero di scorrere lungo un asta metallica. L'asta deve essere posizionata perpendicolarmente alla superficie dell'acqua e deve essere immersa per circa metà della sua lunghezza. Il valore lordo della misura coincide con la distanza tra il galleggiante ed un riferimento situato sulla testa del sensore. Risulta evidente che l'operazione di posizionamento del sensore è critica e deve essere effettuata con estrema precisione, per non mascherare l'accuratezza intrinseca della misura con una scarsa precisione nella determinazione dell'offset. Per facilitare le operazioni di posizionamento sono state ipotizzate alcune soluzioni, tra cui l'aggiunta di una o più aste di prolunga di lunghezza nota, sulla parte superiore dell'unità di misura, in modo da poter definire facilmente l'offset, oppure la predisposizione di un giunto rigido di ancoraggio sullo scafo, inserito già in fase di costruzione dell'imbarcazione, la cui posizione rispetto al piano della coperta sia certificata dal cantiere stesso. Il sistema meccanico di fissaggio deve essere definito univocamente, scegliendo una soluzione che offra sufficiente rigidità ma altrettanta facilità di installazione: sono stati proposti sistemi con ventose ad elevata tenuta da applicare sullo scafo, ma anche staffe fissate in coperta durante le operazioni di misura. Queste tematiche restano aperte per successivi studi.

2.2. Applicazioni del sistema di misura

L'approfondimento delle conoscenze riguardo ai sensori di livello apre le porte ad altre applicazioni. Ad esempio, dalle misure del livello di un singolo sensore inserito in un serbatoio di forma qualunque (ma la cui geometria sia nota), unite ai dati di inclinazione dell'imbarcazione (misurabili con un inclinometro) è possibile risalire alla reale quantità di liquido presente, senza dover ricorrere a sensori multipli collocati in posizioni diverse del serbatoio. Sono in corso studi per verificare la realizzabilità e l'efficienza di un sistema del genere.

Un'altra applicazione, suggerita dall'elevatissima risoluzione dei sensori usati, può essere rivolta a misurare la variazione di peso di un imbarcazione in seguito all'imbarco o allo sbarco di sistemi, macchinari, arredamenti, etc. E' sufficiente collocare l'imbarcazione in cantiere, in una vasca chiusa, e collocare il sensore di livello sul bordo della vasca.

$$\Delta P = S_{imbarcazione} \cdot (\Delta h_{bordolibero}) \cdot \rho_{H_2 0}\,; \quad \Delta h_{bordolibero} = \frac{\Delta P}{S_{imbarcazione} \cdot \rho_{H_2 0}};$$

$$\Delta h_{vasca} = \Delta h_{bordolibero} \cdot \frac{S_{imbarcazione}}{S_{vasca}}; \quad \boxed{\Delta h_{vasca} = \frac{\Delta P}{\rho_{H_2 0} \cdot S_{vasca}};}$$

Nello scrivere le equazioni precedenti è stato ipotizzato che la variazione di livello sia molto piccola e quindi la variazione della superficie dello scafo sul livello dell'acqua (S) sia trascurabile. Spostando il sensore a bordo vasca ci si svincola dalla necessità di conoscere la superficie dello scafo sulla superficie dell'acqua. Con S_{vasca}=200 m^2, si avrà un innalzamento di 50 µm del livello dell'acqua in vasca caricando a bordo un peso di 100 kg. Con questo sistema è possibile arrivare ad apprezzare variazioni di livello di questo ordine di grandezza, e quindi effettuare misure di peso senza ricorrere a nessun sistema di pesatura esterno (quali ad esempio travel-lift dinamometrici, non sempre disponibili per imbarcazioni di grossa stazza).

2.3. Cenni alle comunicazioni wireless Bluetooth

Bluetooth è uno Standard Aperto per telecomunicazioni wireless (trasferimento di dati e voci) a corto raggio. Viene utilizzato per realizzare le Personal Area Network (PAN), cioè

piccole reti dell'estensione tipica di qualche metro, per esempio, reti che collegano computer con varie periferiche (in sostituzione del cavo), reti dati e/o audio in singole abitazioni (home networking, collegamento tra elettrodomestici intelligenti, dispositivi di riscaldamento, condizionamento, intrattenimento, sistemi di allarme).

Bluetooth funziona in banda ISM 2,45 GHz (2400-2483,5 MHz), con una modulazione frequency hopping – spread spectrum (FHSS) per ridurre l'effetto di interferenze e fading (cammini multipli). Sono disponibili 79 canali FHSS. In ogni canale, per minimizzare la complessità del radiotrasmettitore si usa una modulazione binaria FM a 1 Msimbolo/s = 1Mbit/s (lordo). Da standard, la distanza massima di comunicazione è 10 m (100 m con un trasmettitore più potente).

Un ricetrasmettitore Bluetooth è anche detto "dispositivo" bluetooth o "radio" bluetooth. Tipicamente, un canale radio fisico è occupato da più dispositivi sincronizzati su una stessa sequenza di frequency hopping. Tale insieme di dispositivi è chiamato "piconet". Un dispositivo, detto "master", è responsabile della sincronizzazione. Gli altri etichettati come "slave". Lo standard è pensato per essere utilizzato con dispositivi hardware a basso costo (< 10 $ per transceiver). Nelle intenzioni, un collegamento radio Bluetooth non dovrebbe costare più di un equivalente collegamento con cavo (per es. USB).

Riferendosi allo stack dell'ISO-OSI, una rete Bluetooth si basa su diversi livelli. Sopra al canale radio fisico si trovano una serie di canali e collegamenti (link), e i relativi protocolli. In un analisi down-top è possibile elencare il canale radio fisico, che costituisce il livello fisico, a sua volta composto dal canale fisico (physical channel) e dal collegamento fisico (physical link), quindi il livello logico, composto dal trasporto logico (logical transport) e da collegamento logico (logical link), salendo ancora si incontra il livello "L2CAP".

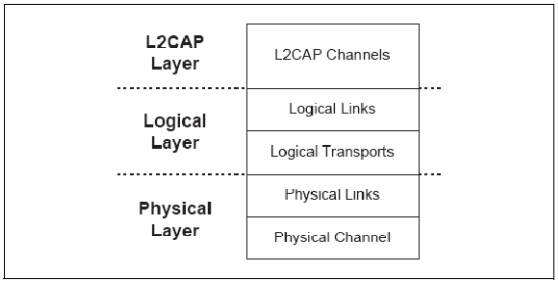


Figura 9. Livelli dello stack Bluetooth

Il livello fisico e buona parte del livello logico sono descritti dalle specifiche della "banda base bluetooth" (bluetooth baseband). Tali livelli non corrispondono esattamente agli strati del modello ISO-OSI. La bluetooth baseband è infatti intermedia tra il livello fisico e il livello data link (figura in basso). Del livello fisico ISO-OSI fanno ovviamente parte le specifiche radio.

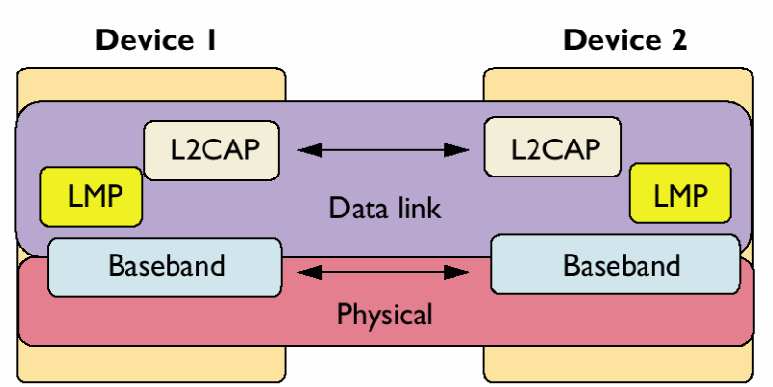


Figura 10. Protocolli dello stack Bluetooth

La parte più alta del livello logico è descritta dal Link Manager Protocol (LMP), che è parte del livello data-link dell'OSI, (di cui fa parte anche il livello L2CAP).

Bluetooth opera in una banda affollata da altri sistemi di comunicazione (la banda ISM è non regolata) quali, ad esempio, le reti WLAN. Per ridurre gli effetti delle interferenze, si deve mettere in atto una tecnica di spread spectrum, che consenta di utilizzare al meglio tutta la banda disponibile (83,5 MHz). In particolare, si implementa una tecnica di "frequency hopping", cioè di variazione della frequenza di trasmissione secondo una particolare sequenza di salto (hopping) pseudocasuale specifica per ciascuna piconet.

Il frequency hopping è una tecnica di cosidetta "collision avoidance". Variando la frequenza di trasmissione su tutta la banda ISM si puo' evitare, con alta probabilità, l'interferenza di altri sistemi di comunicazione a banda stretta fissa o in frequency hopping. In altre parole, si ha collisione solo quando la frequenza di trasmissione si sovrappone alla banda utilizzata dal sistema "interferente", cioè per un intervallo di tempo breve e per una frazione molto piccola del tempo di trasmissione. Le frequenze di trasmissione possibili sono 79, f = 2402 + k MHz, con k=0,...,78. Il tempo è suddiviso in intervalli ("slot") di 625 μs. Durante uno slot viene trasmesso un "pacchetto". La frequenza di trasmissione viene cambiata,

secondo una sequenza pseudo-casuale ogni volta che viene trasmesso un pacchetto. Nel caso tipico in cui un pacchetto ha la durata di uno slot, la frequenza di trasmissione viene cambiata ogni 625 μs ("hopping rate" = 1600 hop/s). La sequenza pseudocasuale di hopping viene determinata dal numero seriale del master.

Lo standard definisce 3 classi di potenza in cui è possibile dividere i dispositivi. La classe 3 corrisponde a una potenza massima del trasmettitore di 1 mW (portata massima del sistema di 10 m); la classe 1 a una potenza massima di 100 mW (portata massima di 100 m). Le antenne (sia trasmettitore, sia ricevitore) sono tipicamente omnidirezionali.

Utilizzando una modulazione binaria di frequenza è possibile realizzare il ricevitore in modo particolarmente semplice anche se, ovviamente, il bit rate è limitato. Il ricevitore può essere supereterodina con bassa frequenza intermedia (~ 3 MHz) o omodina. In tutti e due i casi la sensibilità del ricevitore non è elevata. Lo standard prevede una sensibilità del ricevitore di -70 dBm, per una BER (bit error rate) grezza (cioè escludendo meccanismi di correzione dell'errore) dello 0,1 %. Tipicamente si riesce a realizzare un ricetrasmettitore bluetooth in un unico chip in tecnologia CMOS, e quindi in modo economico.

La rete elementare Bluetooth si chiama **Piconet**. E' costituita da 2 a 8 radiotrasmettitori (**a** e **b** della Figura 11). Il collegamento fisico può essere realizzato solo tra il master e uno slave. Non è possibile avere un collegamento fisico tra due slave.

Da notare che l'architettura della rete è simmetrica, nel senso che ciascun radiotrasmettitore può agire sia da master, sia da slave. Nel momento in cui una rete si forma, il primo dispositivo che partecipa alla rete prende il ruolo di master, gli altri quello di slave.

Un dispositivo può inoltre appartenere a più di una piconet (come slave o come master), ma può essere master solo di una, e in tal caso si chiama "bridge", e permette di unire più piconet in una "scatternet" (in **c** della figura in basso tre piconet sono unite in una scatternet). Un bridge fa parte di piu' piconet a suddivisione di tempo (time-sharing). In questo modo, è in grado di passare informazioni da una piconet all'altra, rendendo lascatternet connessa.

I vari ricetrasmettitori della stessa piconet usano il canale fisico a suddivisione di tempo. Nel caso in cui ad ogni slot viene cambiata la frequenza di trasmissione, il master trasmette negli slot dispari, e lo/gli slave negli slot pari.

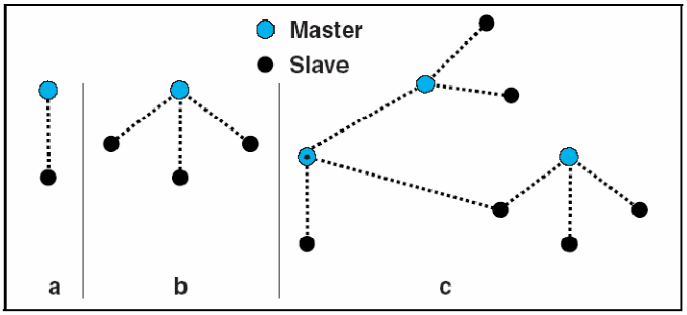


Figura 11. Strutture delle reti Bluetooth

I dati sono trasmessi in pacchetti, ciascuno costituito da tre parti:

- Codice di accesso (access code): 72 bit;
- Intestazione del pacchetto (packet header): 54 bit;
- Payload: da 0 a 2475 bit.

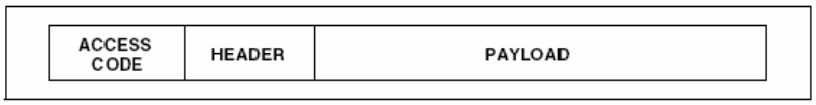


Figura 12. Struttura del pacchetto Bluetooth

Il payload della figura in alto è suddiviso in basso in un payload header, in un carico utile vero e proprio (di nuovo chiamato payload), in una stringa di CRC.

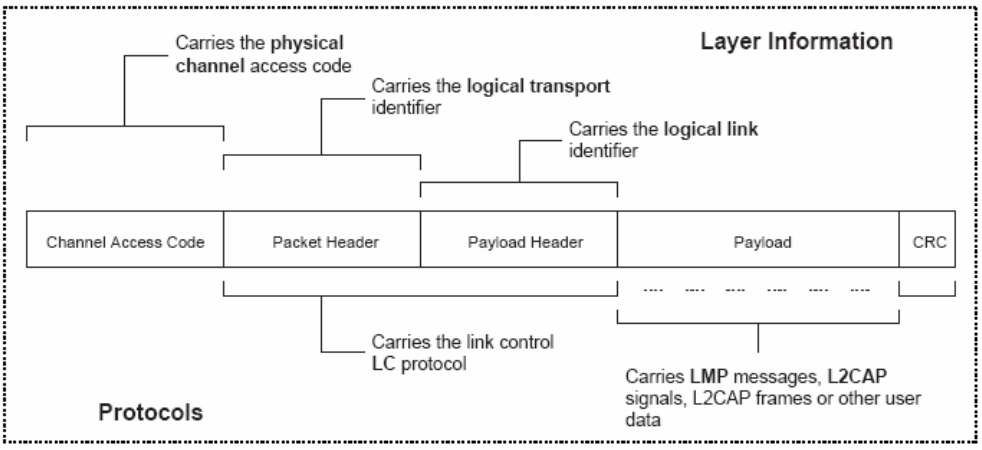


Figura 13. Struttura dettagliata del pacchetto Bluetooth

Sono possibili diversi access code, derivati dal codice identificativo (ID) bluetooth del master o dello slave. Il pacchetto è accettato da un dispositivo nella scatternet solo se l'access code contiene l'ID del master.

Il payload ha un numero di bit variabile in funzione della durata del pacchetto. Il pacchetto puo' durare 1, 3, o 5 slot. Nella figura in basso sono mostrati i tre esempi possibili. In azzurro sono indicati i pacchetti trasmessi dal master. Gli slave possono trasmettere negli slot rappresentati in figura come vuoti. Si ricorda che la frequenza di trasmissione viene variata solo alla fine della trasmissione di un pacchetto intero. Come indicato nella figura in basso, se il pacchetto dura cinque slot, alla fine del pacchetto si salta direttamente alla frequenza prevista dopo 5 slot, in modo da non perdere la sincronia con la sequenza di hopping.

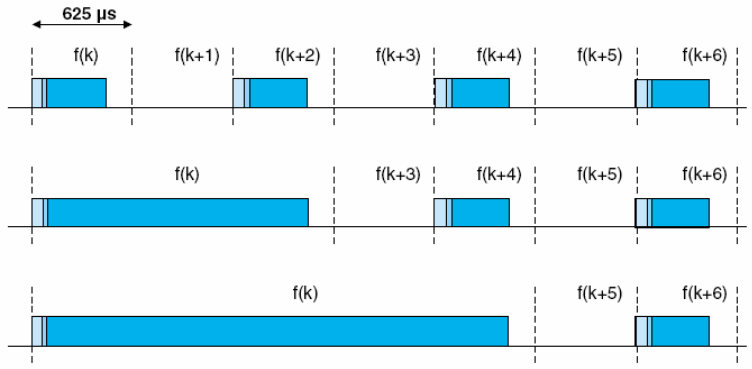


Figura 14. Durata dei pacchetti Bluetooth

Le collisioni sono evitate perché il master decide quale slave deve rispondere, e lo indica nel packet header. Il master ha anche il compito di suddividere la banda disponibile tra i diversi slave.

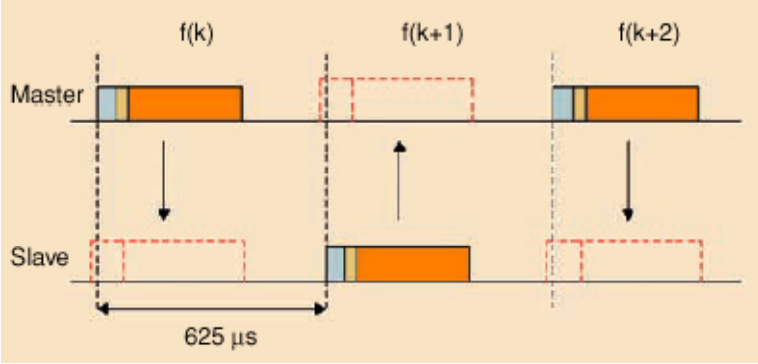


Figura 15. Sequenza di comunicazione

Ogni volta che viene modificata la frequenza vanno persi circa 255 μs dello slot, necessari per la variazione di frequenza. Quindi, se il pacchetto è di uno slot, solo circa 370 μs, sono usati per la trasmissione, corrispondenti a 370 bit. Togliendo i 72 per l'access code e 54 per il packet header, abbiamo circa 244 bit, cioè circa 30 byte.

Bluetooth consente simultaneamente trasmissione voce (comunicazioni bidirezionali asincroni) e trasmissione dati (comunicazioni bidirezionali sincrone) [3]. È possibile realizzare, nei livelli superiori dello stack, diversi tipi di collegamento. Nella rete di misura dei bordi liberi viene sfruttato un livello dello stack in cui è implementato il protocollo SPP (Serial Port Profile): si tratta si un emulazione delle connessioni seriali cablate realizzate con standard RS232. Un dispositivo Bluetooth che utilizza il protocollo SPP presenta all'esterno i segnali tipici di una porta seriale (Tx, Rx, segnali di handshaking). Trasmettendo dati su questa porta, questi vengono presentati sulla porta del dispositivo connesso, come se fosse presente un cavo tra i due.

3. Inclinometro

3.1. Specifiche

E' necessario misurare l'inclinazione della barca sui due assi principali, illustrati in Figura 16.

Per la scelta del sensore di inclinazione adatto a questa applicazione si deve tenere presente che sulle imbarcazioni per cui il sistema è pensato, la massima frequenza dei movimenti di rollio e di beccheggio è di qualche Hertz. Ciò significa che è sufficiente acquisire tali dati alla frequenza di 10 Hz per rispettare il teorema di Nyquist sul campionamento. L'accuratezza necessaria è di 0,5°.

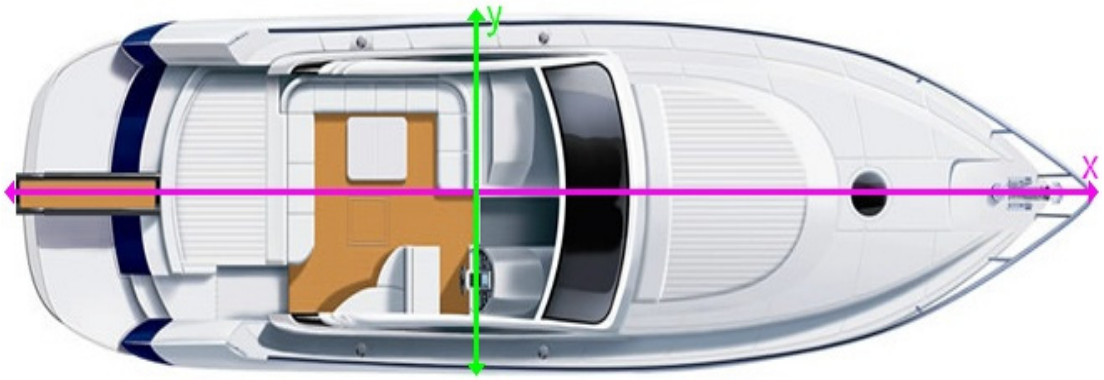


Figura 16. Assi di riferimento per la misura di inclinazione dell'imbarcazione

Deve essere possibile comunicare col dispositivo sia via Bluetooth che via USB.

Il sistema wireless, alimentato a batteria, deve garantire una durata della batteria sufficiente per effettuare diverse prove in mare nella stessa giornata. Si è quindi cercato di ridurre al minimo i consumi ricorrendo alla disattivazione selettiva di porzioni del sistema inutilizzate. Con questi accorgimenti si riesce ad ottenere un consumo massimo di 100mA con dispositivo acceso e situato alla massima distanza di trasmissione (100 m tra inclinometro e PC). In condizioni di utilizzo tipico il consumo è di circa 60mA. Quando il dispositivo è spento assorbe soltanto 3mA: montando una batteria da 1240 mAh il dispositivo impiega più di 15 giorni a scaricarsi completamente quando si trova in standby.

3.2. Tipologia di sensore

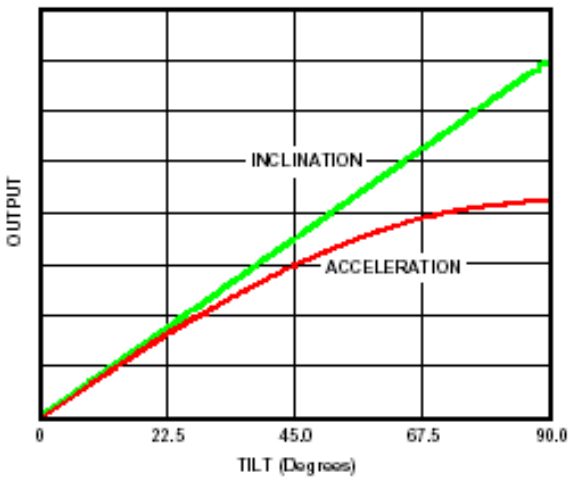
E' stata eseguita un'analisi comparativa tra vari dispositivi elettronici contenenti sensori di inclinazione realizzati con tecniche miniaturizzate di lavorazione del silicio. La scelta è ricaduta sul circuito integrato **ADIS16201** prodotto da *Analog Devices* [4]. Questo è in grado di misurare inclinazione e accelerazione su due assi, *x* e *y*. Il sensore può essere calibrato e configurato attraverso comandi digitali su bus SPI (Serial Peripheral Interface). L'uscita è anch'essa fornita in formato digitale a 12 bit su tale linea di comunicazione. Gli angoli di inclinazione misurabili sono compresi tra ±90°, ed è possibile rilevare accelerazioni fino a ±1,7 g. Le principali caratteristiche del componente sono riportate in Tabella 2.

Input Range (incl.):	± 90° [8]
Input Range (acc.):	± 1,7 g
Risoluzione DAC in uscita (incl.):	12 bit
Range Tensione alimentazione V_{DD}:	3,0 ÷ 3,6 V
Corrente di alimentazione (tip.):	11 mA
Range di Temperatura di lavoro:	-40°C ÷ +115°C
Banda del sensore di accelerazione:	2250 Hz
Dimensioni, Package:	9,2 mm x 9,2 mm x 3,9 mm,

Tabella 2. Caratteristiche principali del chip accelerometro/inclinometro ADIS16201

Il funzionamento del sensore è basato sulla misura dell'accelerazione attraverso una struttura micromeccanica in polisilicio a capacità variabile. L'inclinazione è ricavata in maniera indiretta dalla misura di accelerazione, in modo da calcolare l'angolo rispetto al piano ideale ortogonale all'accelerazione gravitazionale terrestre. Poiché il legame tra le due grandezze è non lineare, l'errore commesso sulla misura di inclinazione è tanto maggiore quanto più ci si avvicina al valore limite di 90°.

[8] Operando nel range ±30° si ottiene un'accuratezza di 0,5°, che è un valore adatto all'applicazione in questione.
[9] Laminate-based Land Grid Array



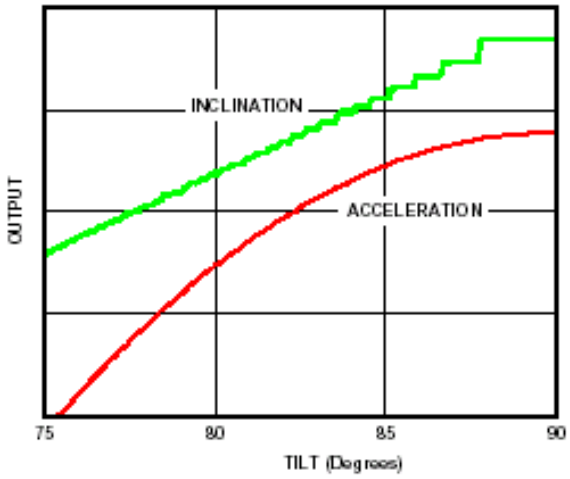


Figure 17 e 18. Relazione non lineare tra inclinazione e accelerazione

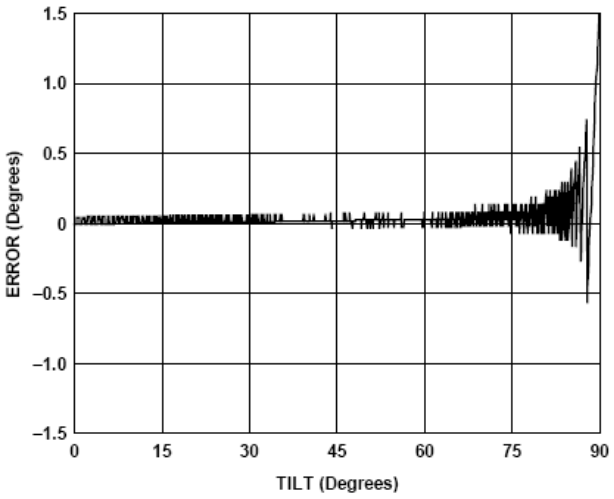


Figura 19. Errore di quantizzazione in funzione dell'inclinazione

I valori di accelerazione misurati sono campionati da un ADC interno e memorizzati, assieme ai valori di inclinazione calcolati, in registri interni, accessibili mediante interfaccia SPI. Ogni operazione di accesso richiede due cicli di trasmissione di 16 bit: nel primo viene trasmesso l'indirizzo del registro da accedere, nel secondo vengono inseriti i dati. La trasmissione è di tipo *MSB first*[10]. Le accelerazioni lungo gli assi *x* e *y* sono rappresentate con codifica in complemento a due.

[10] MSB (Most Significant Byte) First: il primo byte trasmesso è il più significativo del dato.

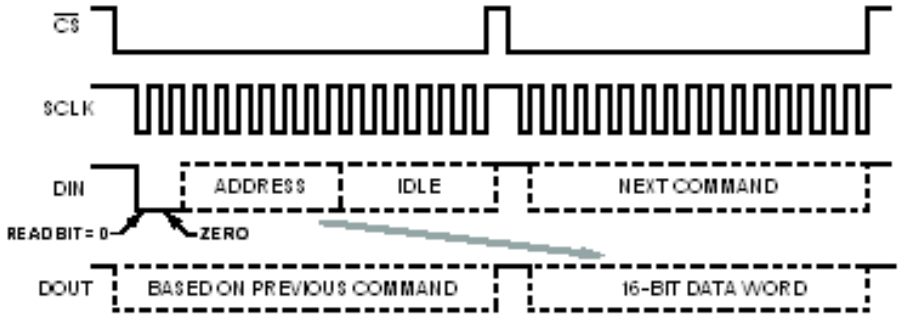


Figura 20. Temporizzazione di accesso per la lettura dati

DIN	W / R	0	A5	A4	A3	A2	A1	A0	x	x	x	x	x	x	x	x
DOUT	ND	EA	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
	Upper Byte								Lower Byte							

Figura 21. Struttura dettagliata dei comandi di lettura.

Il chip ADIS16201 contiene anche una serie di registri attraverso la cui configurazione è possibile settare particolari modalità di funzionamento. Agendo sul registro *COMMAND* è possibile operare sul chip un azzeramento dei dati in uscita (a scopo di calibrazione) oppure inviare un comando di *Reset Software*, che resetta, o ancora eseguire un *Factory Reset*, che riporta i registri dati ai valori che corrispondono alla calibrazione effettuata dal costruttore. È possibile attivare un filtraggio passa-basso sui dati effettuando un'operazione di media sui risultati. L'entità dell'operazione, che corrisponde al numero di campioni su cui si effettua la media, si imposta scrivendo un valore da 00_H a 08_H nel registro *AVG_CNT*[11]. Sono impostate anche funzioni di allarme per segnalare particolari condizioni di funzionamento o problemi nel sistema.

L'ADIS16201 viene prodotto esclusivamente in un package a montaggio superficiale i cui pin sono situati sotto al case stesso (LGA16, Land Grid Array). Tale package può essere saldato solo mediante processi industriali. Per la prototipizzazione e la piccola serie si è scelto di utilizzare una versione dello stesso sensore (fornita sempre da Analog Device) già montata su PCB e connessa alla scheda principale tramite due connettori.

[11] Numero dei valori su cui si effettua la media = 2^{AVG_CNT}.

3.3. Descrizione del sistema

DESCRIZIONE DELLA MECCANICA

Per questo dispositivo si è scelto di contenere il più possibile peso e ingombri esterni, ed allo stesso tempo garantire un grado di protezione adeguato (IP54) all'elettronica. Linvolucro scelto è costruito da Hammond Manufacturing (articolo 1594B): si tratta di una scatola in ABS, corredata di una guarnizione in gomma che viene serrata tra i bordi del coperchio e quelli della scatola. Mediante un pulsante (ITW Switches 76-5910R), dotato di indicatore luminoso a LED, vengono effettuate le operazioni di accensione e spegnimento. La presenza del LED integrato consente di ridurre ulteriormente l'ingombro. Sulla scatola è inoltre montato un connettore di tipo mini-USB (Bulgin PX0443) per consentire la connessione USB e la ricarica della batteria interna. Sia il pulsante che il connettore offrono un grado di protezione superiore a quello della scatola.

DESCRIZIONE DELL'ELETTRONICA

In Figura 22 è riportato uno schema generale che illustra i vari moduli del dispositivo. Di seguito si trova la descrizione dettagliata di ognuno di essi.

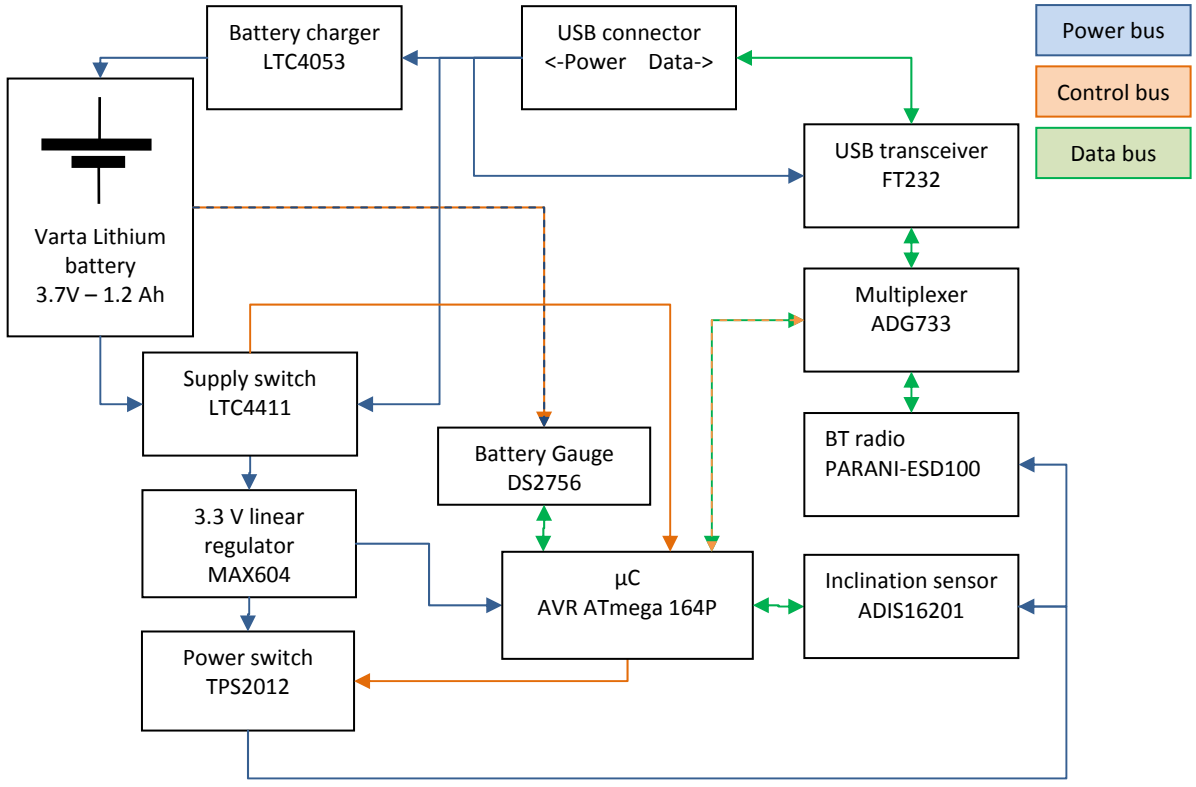


Figura 22. Schema generale dell'inclinometro

Il dispositivo è progettato in modo da rendere possibile sia il funzionamento via USB, sia la modalità wireless, via Bluetooth. In questa ottica viene effettuata la scelta automatica tra alimentazione da batteria o da USB a seconda che il connettore USB (che fornisce una tensione di 5 V) sia inserito o meno.

Con connettore disinserito e dispositivo spento, la radio Bluetooth e il sensore di inclinazione non sono alimentati e la corrente assorbita dalla batteria è inferiore a 4 mA. Alla pressione del tasto presente sull'involucro del dispositivo viene attivata la radio Bluetooth Sena Parani ESD-100, Classe 1 (U17) [5], mediante il microcontrollore ATtiny2313 (U3).

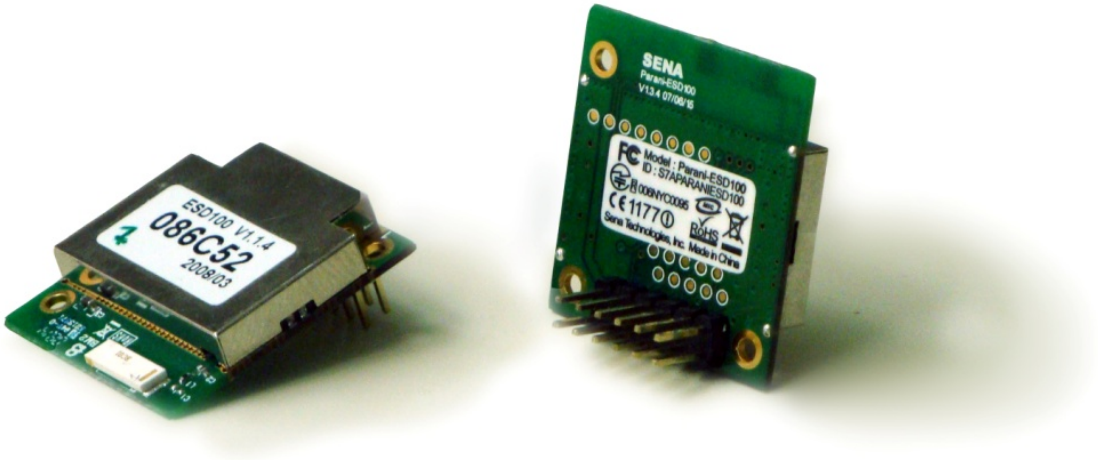


Figura 23. Il modulo Bluetooth-seriale Sena Parani ESD-100

L'operazione di accensione è segnalata da un opportuno lampeggio del LED integrato nel pulsante (comandato dall'uscita *LED_driving* del µC), dipendente dallo stato di carica della batteria. Si avrà un lampeggio lento in condizioni di batteria carica, veloce quando la carica della batteria è inferiore al 25 % (ovvero circa 300mAh, con cui si stimano 6 h di funzionamento continuo o 150 ore di standby). In questo modo si comunica all'utente la necessità di una ricarica anche se non viene effettuato il collegamento con il PC.

L'attivazione della radio Bluetooth avviene imponendo un livello basso sull'uscita */POWERDOWN* del µC, che rende attivo il collegamento della tensione di alimentazione con U16 attraverso lo switch high-side U20 (TPS2012D). Allo stesso tempo viene alimentato anche l'accelerometro ADIS16201.

Il microcontrollore viene collegato alla radio Bluetooth attraverso il commutatore elettronico ADG733 (U16). Quando un PC si connette al dispositivo, viene segnalato l'avvenuto collegamento con opportuno lampeggiamento del LED.

Con connettore inserito si provvede alla carica della batteria al Litio. La carica è effettuata dal chip LTC4053 (U14), che porta la batteria fino a 4,2 V a partire da un'alimentazione a 5 V. Si prevede di poter caricare la batteria con 3 diversi possibili valori di corrente, 100 mA, 300 mA, 500 mA, a seconda dei casi in cui ci troviamo (dipende dalla sorgente che fornisce l'alimentazione USB, per ottenere 500 mA è ad esempio necessario disporre di un HUB alimentato). Il chip LTC4053 infatti dispone di un pin *PROG* sul quale, nel funzionamento normale, si trovano 1,5 V. La corrente di carica è generata da uno specchio di corrente con molteplicità 100: si ottiene così una corrente di ricarica 100 volte più grande di quella che scorre in un resistore connesso tra tale pin e massa: in questo modo è possibile impostare la corrente di carica semplicemente variando tale resistenza.

Vale la seguente relazione : $I_{CHG} = (1{,}5\ V \times 1000) / R_{PROG}$, quindi invertendo $R_{PROG} = 1500\ V / I_{CHG}$.

Perciò occorrono

$$I_{CHG} = 100\ mA \Rightarrow R_{PROG} = 15\ k\Omega;$$

$$I_{CHG} = 300\ mA \Rightarrow R_{PROG} = 5\ k\Omega;$$

$$I_{CHG} = 500\ mA \Rightarrow R_{PROG} = 3\ k\Omega.$$

Questi valori di R equivalente si ottengono mettendo in parallelo alternativamente tre resistenze da 15 kΩ, 7,68 kΩ, 7,68 kΩ (valori commerciali⇒7,68 kΩ anziché 7,5 kΩ). Le due da 7,68 kΩ sono connesse a due pin del µC (PD4, PD5): via firmware si pongono a massa mettendo i resistori in parallelo agli altri, oppure si pongono in alta impedenza, rendendole inefficaci ai fini della determinazione della corrente di ricarica. In pratica la carica a 300 mA non viene mai utilizzata e non è implementata nel firmware.

La carica viene controllata da un timer il cui periodo è impostato dalla costante di tempo di un circuito RC nel quale il condensatore è connesso esternamente. Allo scadere del timer la ricarica viene arrestata, per essere riavviata in seguito, quando la tensione della batteria

scende sotto 4,05 V. Si è impostato il periodo di ricarica a 3 ore, utilizzando un condensatore da 100 nF. I componenti che completano la configurazione circuitale del charger seguono lo schema consigliato dal costruttore. In particolare il pin di *shutdown* è posto a Vcc disabilitando questa funzione. E' stato inoltre omesso il controllo termico della batteria, tenuto conto delle esigue correnti che vengono assorbite da essa [6].

Con il *battery gauge* DS2751 si effettua il controllo dello stato di carica della batteria. L'informazione viene trasmessa al µC attraverso interfaccia **1-wire** (pin *GaugeData* su PB4 del µC):

> Questo standard di comunicazione half-duplex ha come obiettivo la riduzione estrema del numero di collegamenti, a discapito della velocità di trasmissione. Come il nome stesso suggerisce, viene utilizzato un unico collegamento tra due dispositivi. Per dispositivi che richiedono esigue correnti di alimentazione, la linea dati può essere la stessa che alimenta il dispositivo. Tale linea è mantenuta a livello logico alto da un pull-up.

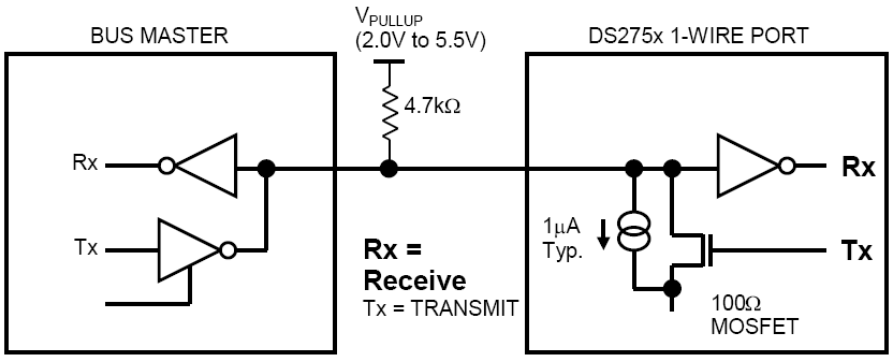


Figura 24. Comunicazione One-Wire

> Quando avviene una trasmissione di dati la linea viene portata al livello logico basso per brevi periodi. L'informazione che viaggia sulla linea è contenuta nella ***durata*** degli impulsi di livello logico basso. Anche in questo caso tra i dispositivi collegati alla linea deve essere definito un *master*. Questo inizia la comunicazione resettando tutti gli altri dispositivi imponendo uno zero logico sulla linea per 480 µs. Questo tempo è sufficiente per privare dell'alimentazione quei dispositivi che sono alimentati dalla linea stessa. Quindi inizia la trasmissione vera e propria da parte del *master*: un 1 logico è codificato

come un impulso basso di durata inferiore a 15 µs, uno 0 logico consiste in un impulso basso di 60 µs. I dispositivi *slave* possono interpretare l'informazione campionando la linea dopo 30 µs dall'inizio dell'impulso. Quando il *master* deve ricevere dati deve comunque generare un impulso basso di 1 µs: il dispositivo *slave* che sta trasmettendo manterrà la linea bassa per 60 µs quando deve essere trasmesso 0, mentre non forzerà la linea (che sarà riportata a livello logico alto dal pull-up) quando trasmette 1. Il master leggerà il dato campionando la linea dopo 2 µs dall'inizio dell'impulso. In questo modo un'unica linea è usata per alimentazione, dati e sincronismo. La realizzazione di linee multi-drop (con più dispositivi *slave*) è possibile facendo ricorso ad un arbitraggio basato sugli *ID* dei singoli *slave*. [7]

Il battery fuel-gauge viene interrogato automaticamente all'accensione del dispositivo, per scegliere la segnalazione luminosa opportuna (descritta in precedenza). Inoltre è possibile richiedere informazioni sullo stato della batteria anche tramite l'interfaccia presente sul PC.

Per effettuare la ricarica della batteria anche quando il dispositivo è acceso, è necessario che questa venga scollegata elettronicamente dal resto del circuito. In caso contrario, parte della corrente di ricarica erogata dal caricabatterie sarebbe utilizzata per alimentare il dispositivo stesso e non contribuirebbe alla carica della batteria.

Per aggirare questo problema si utilizza un opportuno collegamento tra le alimentazioni, nel quale si sfrutta un diodo Schottky (D1) e il diodo ideale LTC4411 (U10).

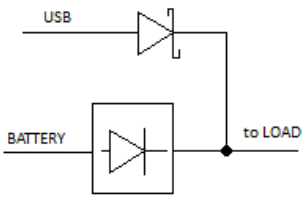


Figura 25. Collegamento della batteria

La caduta ai capi del diodo ideale è pressoché nulla: quando si connette l'USB il diodo ideale si interdice poiché:

$$V_{U11} = V_{batt} - \left(V_{USB} - V_{\gamma_{D1}}\right) \cong 3{,}7\ V - \ (5\ V - 0{,}4\ V) = -0{,}9\ V$$

La batteria risulta pertanto disconnessa dal carico. Quando U11 si interdice la sua uscita *STAT* viene portata in alta impedenza e quindi innalzata a VCC con un pull-up resistivo (invece il pin è collegato a massa quando il diodo conduce). Il µC può quindi conoscere lo stato dell'alimentazione semplicemente leggendo il valore logico di questo pin.

Il collegamento dell'alimentazione USB alimenta il transceiver USB-USART FTDI FT232BM (U11). Questa operazione non è però sufficiente per poter instaurare la comunicazione USB. Infatti è previsto di poter ricaricare l'inclinometro con un adattatore da parete o da presa accendisigari, e allo stesso tempo comunicare tramite rete wireless. Quando invece il dispositivo è collegato direttamente ad una porta USB del PC e si intende stabilire una connessione tramite cavo, è necessario agire sul pulsante dell'inclinometro. Il µC pone a *SEL_Comm* a livello logico alto, collegando i pin RXD e TXD del µC al transceiver USB attraverso il commutatore elettronico U16. La comunicazione via USB può quindi avere luogo.

Nelle successive versioni di questa scheda è necessario inserire una memoria e^2prom collegata al transceiver USB, come descritto nel datasheet [8]. In questo modo è possibile assegnare al dispositivo un identificativo che viene visualizzato automaticamente quando si effettua la connessione USB (esattamente come avviene durante la ricerca dei dispositivi Bluetooth).

Le connessioni tra il sensore di accelerazione ADIS16201 (U13) e microcontrollore sfruttano le interfacce hardware di quest'ultimo per realizzare una comunicazione **SPI (Serial Peripheral Interface)** o **3-Wire**:

> Si tratta di un protocollo di comunicazione inter-chip sincrono e full-duplex. É necessario che tutti i dispositivi condividano lo stesso clock. Un interfaccia SPI consiste in un registro a scorrimento con ingresso ed uscita seriali disponibili all'esterno.
>
> Nel caso in cui i dispositivi che devono comunicare siano soltanto due, uno verrà definito come *master* (ovvero avrà il compito di generare il segnale di sincronismo), l'altro come *slave* (utilizzerà il clock generato dal master). I dispositivi sono connessi con almeno tre segnali, come illustrato nella figura seguente.

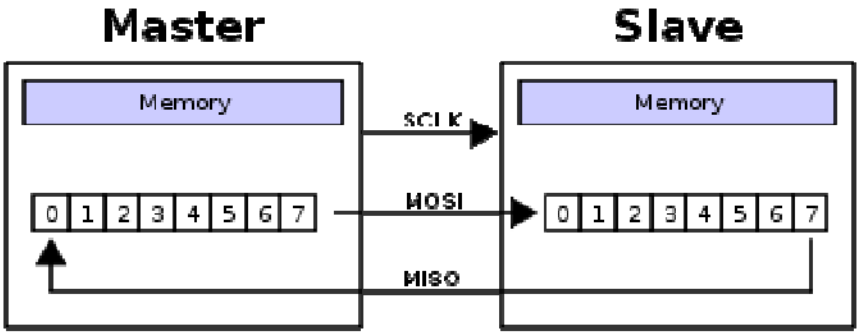


Figura 26. Comunicazione SPI

- SCLK — Serial Clock (output from master);
- MOSI/SIMO — Master Output, Slave Input (output from master);
- MISO/SOMI — Master Input, Slave Output (output from slave).

I due registri sono connessi in modo da formare un registro ad anello. Per trasferire una parola questa deve essere caricata nel registro a scorrimento. Sia *master* che *slave* possono caricare il proprio registro. Quindi il *master* genera un treno di *n* impulsi di clock, dove *n* è la lunghezza di ogni registro. In questo modo il registro ad anello viene fatto scorrere di *n* posizioni: il contenuto del registro del *master* viene trasferito nel registro dello *slave* e **contemporaneamente** i bit caricati nel registro dello *slave* sono spostati in quello del *master*. A questo punto ogni dispositivo può accedere al proprio registro con una lettura parallela.

Questo semplice esempio può essere esteso al caso in cui si abbiano più dispositivi *slave*, per formare un ***BusSPI***. In Figura 27 sono schematizzati due metodi di realizzazione del bus.

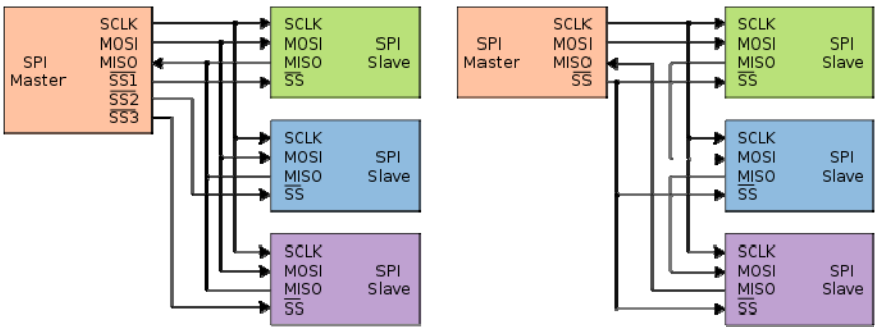


Figura 27. Connessioni Bus-SPI: independent-slave e daisy-chain

Nel primo tutti gli *slave* sono connessi in parallelo ma il *master* ne abilita uno alla volta, agendo sul pin di *SlaveSelect* (o *Chip Select*) di ogni dispositivo: sono necessari *3+k* collegamenti. Nel secondo caso tutti i registri i *k slave* sono connessi in serie, a formare un unico registro ad anello. Tutti gli *slave* sono abilitati contemporaneamente. In questo caso sono necessari solo 3 collegamenti (i pin di *Slave Select* possono essere lasciati sempre attivi) ma per ottenere la parola generata dal primo dispositivo *slave* della catena sono necessari *k* treni di impulsi di clock. [9]

Nel nostro caso il µC agisce da master e il sensore è l'unico slave. Non esistono quindi problemi di arbitraggio e il pin di *chip-select* del sensore di inclinazione è stato collegato direttamente a massa.

L'alimentazione per tutti i chip è fornita dal regolatore MAX604 (U10) che regola a 3,3 V la tensione d'ingresso da batteria o USB. Per l'FTDI sono utilizzate due alimentazioni: la 5 V fornita dal connettore alimenta la sezione USB del chip, modo da disattivarsi (quindi ridurre al minimo il consumo) quando il connettore USB è disinserito. Per la VCC-IO di tale chip (alimentazione dell'interfaccia USART) si usa invece la 3,3 V regolata come suggerito dal costruttore.

Il pulsante è connesso all'ingresso PD2 del µC, che è selezionabile come sorgente di interruzione esterna. Sebbene l'inclinometro sia dotato di una funzione di spegnimento automatico dopo 10 minuti di inattività, è prevista la possibilità di spegnere forzatamente il dispositivo tramite la pressione prolungata del pulsante. Il LED rosso presente al centro del pulsante è pilotato dall'uscita PB1 del µC. Per sfruttarne al massimo la luminosità viene pilotato con una corrente di 20mA: si utilizza quindi una resistenza di limitazione.

$$V_{LED} = 2{,}2\ V;\ R_{LED} = \frac{(3{,}3\ V - 2{,}2\ V)}{20\ mA} = 55\ \Omega \cong 56\ \Omega$$

Per lo switch high-side TPS2012D (U20) si adotta la configurazione circuitale consigliata dal costruttore (vedi data-sheet). [10]

Sono state inserite capacità di filtraggio sull'alimentazione, in prossimità dei pin di alimentazione dei vari chip, uniformando i valori degli elettrolitici a 4.7 µF e a 100 nF per i ceramici.

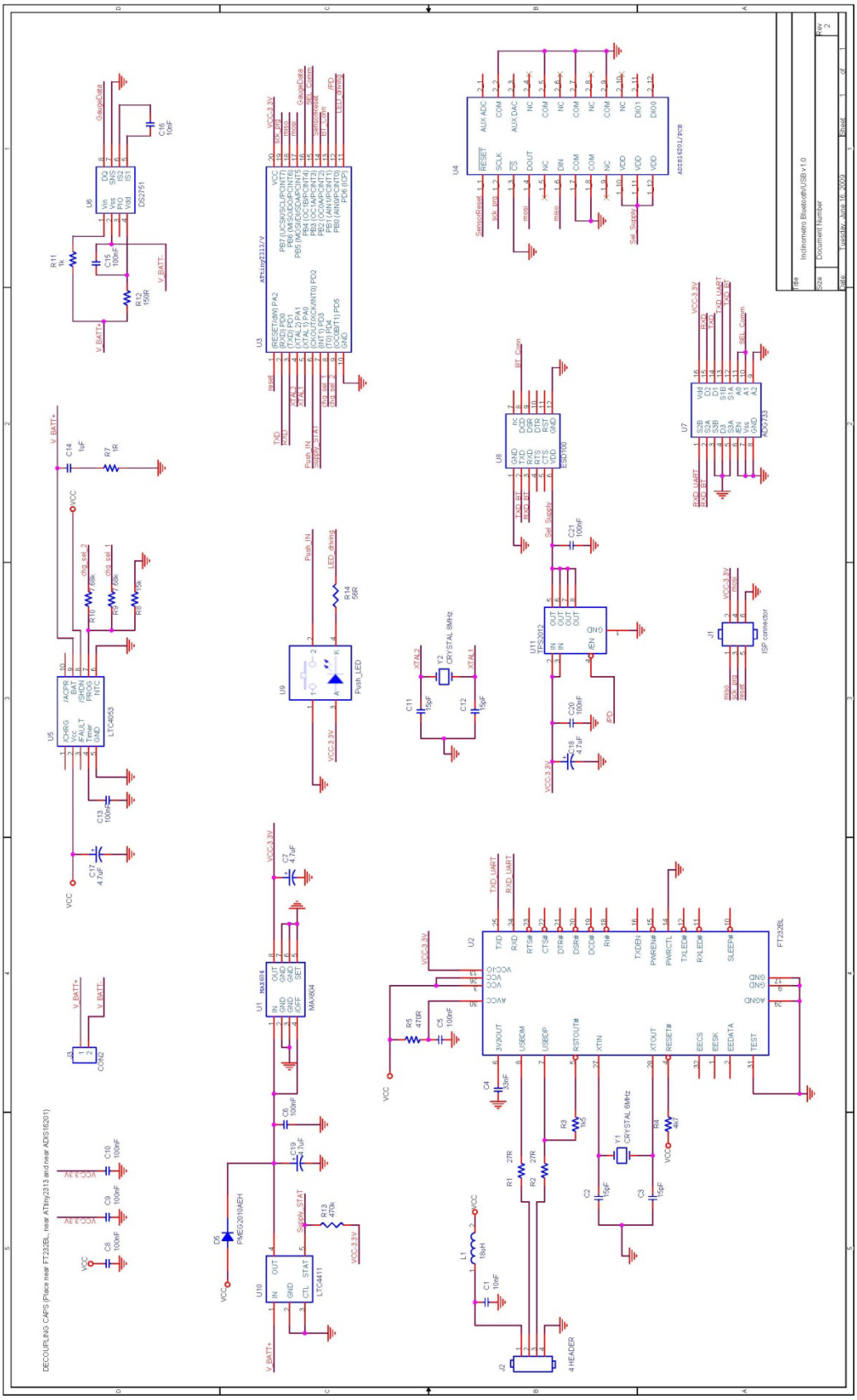


Figura 28. Schema elettrico dell'inclinometro Bluetooth

DESCRIZIONE DEL FIRMWARE

Il firmware è stato scritto in C, utilizzando le librerie AVR-libc all'interno dell'ambiente di sviluppo AVR Studio. Il uC implementa la macchina a stati illustrata in Figura 29.

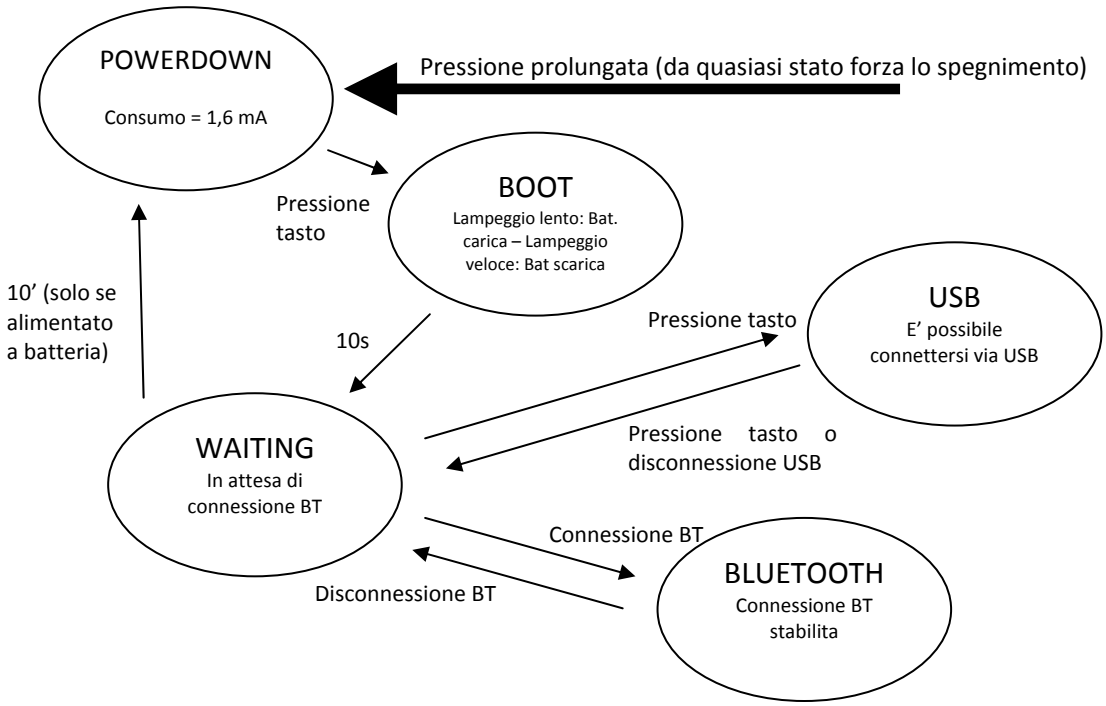


Figura 29. Macchina a stati dell'inclinometro Bluetooth

Il microcontrollore effettua le operazioni in un ciclo continuo, eseguito ogni 10 ms (la sincronizzazione avviene tramite interrupt, sfruttando la struttura hardware di Timer0). Dopo aver eseguito le operazioni di servizio e di aggiornamento dello stato, il µC gestisce la ricezione e l'invio di dati da/verso il PC, tramite interfaccia seriale. La figura seguente schematizza il meccanismo di sincronizzazione tramite *semaforo* (Figura 30).

La ISR (Interrupt Service Routine) del timer provvede inoltre ad incrementare il registro *time*, che viene utilizzato per decidere quando attivare la procedura di spegnimento per timeout.

Anche la gestione della pressione del tasto (breve o prolungata) avviene tramite un flag settato dalla ISR delle interruzioni esterne, e resettato dalla funzione principale del programma, nel momento in cui acquisisce l'informazione.

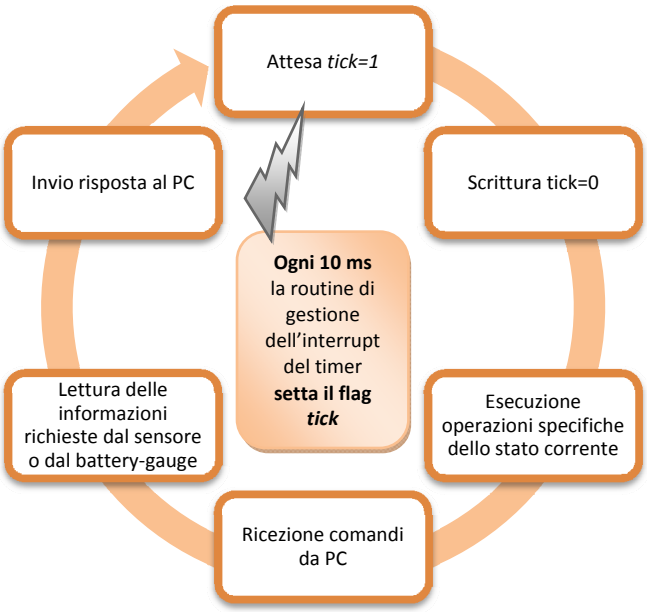


Figura 30. Struttura del firmware

La macchina a stati finiti è implementata semplicemente definendo una variabile di tipo *enumerato*:

```
enum state {POWERDOWN,USB,WAITING,BLUETOOTH,BOOT} state_var;
```

Quindi, durante l'esecuzione del programma si entra in un costrutto *switch* che definisce le operazioni da effettuare in ogni stato attraverso la keyword *case*:

```
switch (state_var) {
                   //POWERDOWN state
                   case POWERDOWN: operazioni; break;

                   //BOOT state
                   case BOOT: operazioni; break;

                   etc…
                   }
```

Per gestire il lampeggio del LED sono state scritte alcune funzioni apposite.

Si utilizza un contatore, incrementato da un apposita funzione che viene chiamata ad ogni aggiornamento della macchina a stati. Questo contatore viene resettato all'interno della stessa funzione quando raggiunge un valore limite precedentemente impostato (**b** in Figura 31, delimita l'intervallo blu): tale valore misura il periodo di ripetizione del lampeggio in quanti di tempo (in questo caso di 10 ms ognuno). L'accensione e lo spegnimento del LED

vengono effettuati comparando il valore del contatore con quello contenuto in un registro (**a** in Figura 31, delimita l'intervallo rosso): se il contatore contiene un valore minore di **a** il LED viene acceso, altrimenti viene spento.

Utilizzando tecniche simili è possibile realizzare lampeggi più complicati utilizzando altri registri in modo da ottenere dei pattern di *n* lampeggi che si ripetono con un periodo maggiore del tempo che intercorre tra un lampo ed il successivo. Questo consente ad esempio di far lampeggiare il LED a gruppi di *k* lampi. Nell'applicazione in esame non è stata utilizzata questa funzione.

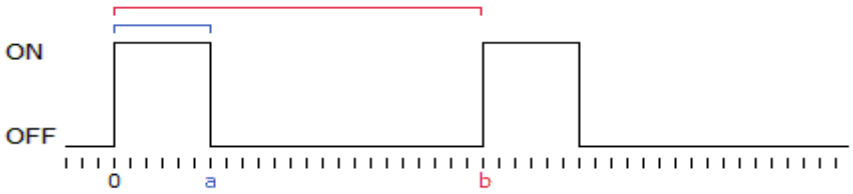


Figura 31. Lampeggio del LED (inclinometro)

Tutti i registri utilizzati vengono settati con una apposita funzione, chiamata ogni volta che avviene un cambio di stato (e quindi un cambio di lampeggio) e alla quale vengono passati i valori da assegnare ai registri.

Le segnalazioni luminose emesse dal dispositivo sono elencate in Tabella 3.

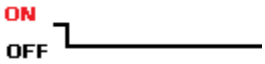
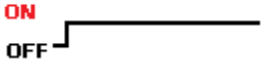
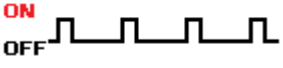
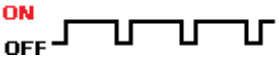
Stato di funzionamento		Attività Led
Power-Down		
Boot	*Low Battery*	
	Battery OK	
Waiting Connection		
Bluetooth COMM		
USB COMM		

Tabella 3. Segnalazioni luminose dell'inclinometro

Per ricevere i comandi dal PC, la radio Bluetooth e il transceiver FTDI, sono stati collegati, per mezzo si un commutatore, all'interfaccia USART del microcontrollore. Una volta settati i registri interni in modo da configurare correttamente la comunicazione (trasmissione asincrona a 115200 bps, 8n1), questa è gestita in maniera completamente trasparente per il programmatore: quando viene ricevuto un dato viene reso disponibile sul registro *UDR* e viene settato il flag *RXC* (è anche possibile far generare un interrupt in corrispondenza di ogni ricezione). Per inviare un byte è sufficiente assicurarsi che il registro *UDR* sia libero (controllando il flag *UDRE*), quindi scrivere il dato sul registro *UDR*.

E' possibile agire sul dispositivo con un set di comandi in formato ASCII.

Comando ASCII [HEX]	Operazione	Risposta e decodifica
1 [31]	Lettura tensione batteria	2 Byte: S+10 bit Left Justified – 4,88mV/bit
2 [32]	Lettura corrente batteria	2 Byte: S+12 bit Left Justified – 0,78125 mA/bit
3 [33]	Lettura carica batteria	2 Byte: S+15 bit Left Justified – 312,5 µAh/bit
4 [34]	Lettura inclinazione	2 Byte X + 2 Byte Y: 2's complement 0,1°/bit
5 [35]	Lettura accelerazione	2 Byte X + 2 Byte Y: 2's complement 0,4625 mg/bit
6 [36]	Impostazione corrente di ricarica:	Nessuna risposta
7 [37]	Impostazione corrente di ricarica:	Nessuna risposta
8 [38]	Lettura stato del sensore di inclinazione	Si veda [1], pag. 25
D [44]	Abilitazione modalità di scarica rapida	Nessuna risposta
E [45]	Azzeramento carica della batteria	Nessuna risposta
V [56]	Lettura versione del firmware	2 byte (unsigned 16 bit integer, MSB first)

Tabella 4. Comandi dell'inclinometro Bluetooth

Il comando "7" imposta la corrente di ricarica al valore di 500 mA. Quando si spegne l'inclinometro la corrente di ricarica viene automaticamente reimpostata a 100 mA: questo per evitare che il dispositivo assorba correnti elevate da una sorgente di alimentazione che non può erogare 500 mA (per cui è necessario inviare il comando di abilitazione ogni volta che si accende il dispositivo).

Il comando "D" inibisce la transizione nello stato *POWERDOWN* quando scade il timeout: in questo modo è possibile scaricare più velocemente la batteria per effettuare l'operazione di **taratura**. Questa è eseguita scaricando completamente la batteria fin quando il dispositivo non si spegne. Quindi viene iniziata la ricarica della batteria, monitorandone continuamente

la tensione. Quando questa raggiunge il valore di 3,2 V il registro di carica presente nel battery fuel-gauge viene resettato inviando al dispositivo il comando “E”.

La comunicazione SPI col sensore di accelerazione è stata implementata ricorrendo all’interfaccia hardware USI (Universal Serial Interface) presente sul microcontrollore, configurata nel modo suggerito dal costruttore [11], mentre per gestire il protocollo 1-wire (battery fuel-gauge) si fa ricorso a funzioni scritte direttamente nel firmware del µC, usando routine di ritardo per definire con precisione le temporizzazioni di lettura e scrittura.

3.4. Realizzazione del PCB e collaudo

Il progetto geometrico e la realizzazione della scheda non presentano particolari vincoli di layout tranne le dimensioni, tali da permettere al PCB di essere alloggiato all’interno dell’apposita scatola. Sono stati previsti degli scassi circolari per il posizionamento delle viti di fissaggio del coperchio ed uno al centro per consentire un montaggio compatto del pulsante, il cui corpo penetra attraverso la scheda stessa. L’unico dettaglio cui è necessario prestare attenzione è il collegamento della *Rsense* del battery fuel-gauge (U6) in modo tale che le piste che collegano i pin di *Sense* del chip alla resistenza stessa non siano attraversate da corrente e quindi non contribuiscono alla differenza di potenziale misurata, come in una misura di resistenza a 4 fili. A questo proposito, si veda la sezione **b** della figura seguente: in questo caso le piste in verde sono attraversate dalla corrente di alimentazione di tutto il dispositivo pertanto generano una caduta di tensione non trascurabile. Il montaggio rappresentato in **c** è invece corretto.

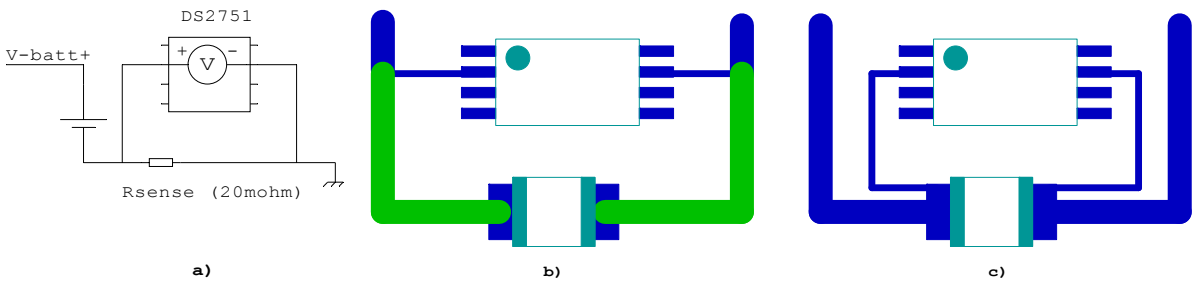


Figura 32. Collegamento di *Rsense*: a) schema elettrico; b) layout errato; c) layout corretto

Per la realizzazione della scheda ci si è rivolti ad un’azienda esterna: PCB-Proto realizza circuiti stampati fino a 6 strati, e fornisce servizi quali serigrafia, deposizione di solder-mask, metallizzazione dei fori, fresatura. Vengono offerti flussi di processo con caratteristiche,

tempi di realizzazione e costi diversi: è stato scelto il servizio *Hobby Pool* con il quale possono essere realizzate da 2 a 50 schede a singola o doppia faccia, con solder-mask su entrambi i lati e serigrafia su un singolo lato.

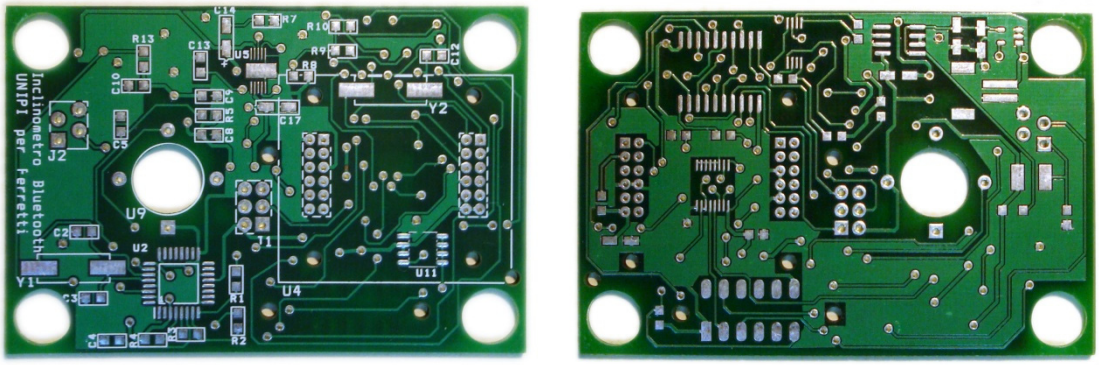


Figura 33. PCB dell'inclinometro

Il montaggio e la saldatura dei componenti sono stati effettuati manualmente. In fase di collaudo del firmware si è reso necessario effettuare delle modifiche al routing: questo è stato possibile interrompendo meccanicamente le piste esistenti ed realizzando i nuovi collegamenti con dei fili. Le modifiche effettuate sono quindi state integrate nel progetto schematico e geometrico per le successive realizzazioni della scheda.

La programmazione del microcontrollore avviene direttamente sulla scheda, attraverso l'apposita interfaccia SPI. A questo scopo è stato previsto un connettore per collegare il dispositivo alla scheda di sviluppo Atmel STK500: questa può essere collegata alla porta seriale di un PC da cui si può caricare il firmware sul µC.

Il firmware è stato testato per parti: inizialmente è stato caricato esclusivamente il bootstrap, la macchina a stati e la routine di lampeggio del LED. Verificata la correttezza delle transizioni di stato è stata aggiunta la sezione di ricezione dei comandi di inclinazione e la lettura del registro dal sensore di inclinazione. L'ultima parte di programma inserita è quella che gestisce la comunicazione con il battery fuel-gauge.

3.5. Interfaccia LabVIEW

Come già accennato, questo inclinometro è stato inizialmente costruito per essere collegato ad un PC su cui gira un applicativo sviluppato presso il Dipartimento di Ingegneria

dell'Informazione dell'Università di Pisa, in grado di coadiuvare i collaudatori di imbarcazioni da diporto durante le prove in mare (BlackBox, vedi paragrafo 1.3). In questo software è presente una sezione che acquisisce dati dall'inclinometro, li visualizza ed effettua un data logging.

Figura 34. Visualizzazione dati di inclinazione in BlackBox

Durante il collaudo del dispositivo sono state sviluppate altre applicazioni che permettono all'utente di gestire direttamente l'inclinometro, da un livello più vicino all'hardware stesso.

In particolare sono disponibili due software per la verifica del dispositivo: il primo, ***Inclinometer Tester***, permette di acquisire i dati di inclinazione, accelerazione e stato della batteria e di visualizzarli a schermo; il secondo, ***Inclinometer Battery Manager***, consente di controllare lo stato della batteria ed intervenire su di esso, ad esempio automatizzando la procedura di taratura. Nelle figure successive sono presentati alcuni screen-shot delle due applicazioni.

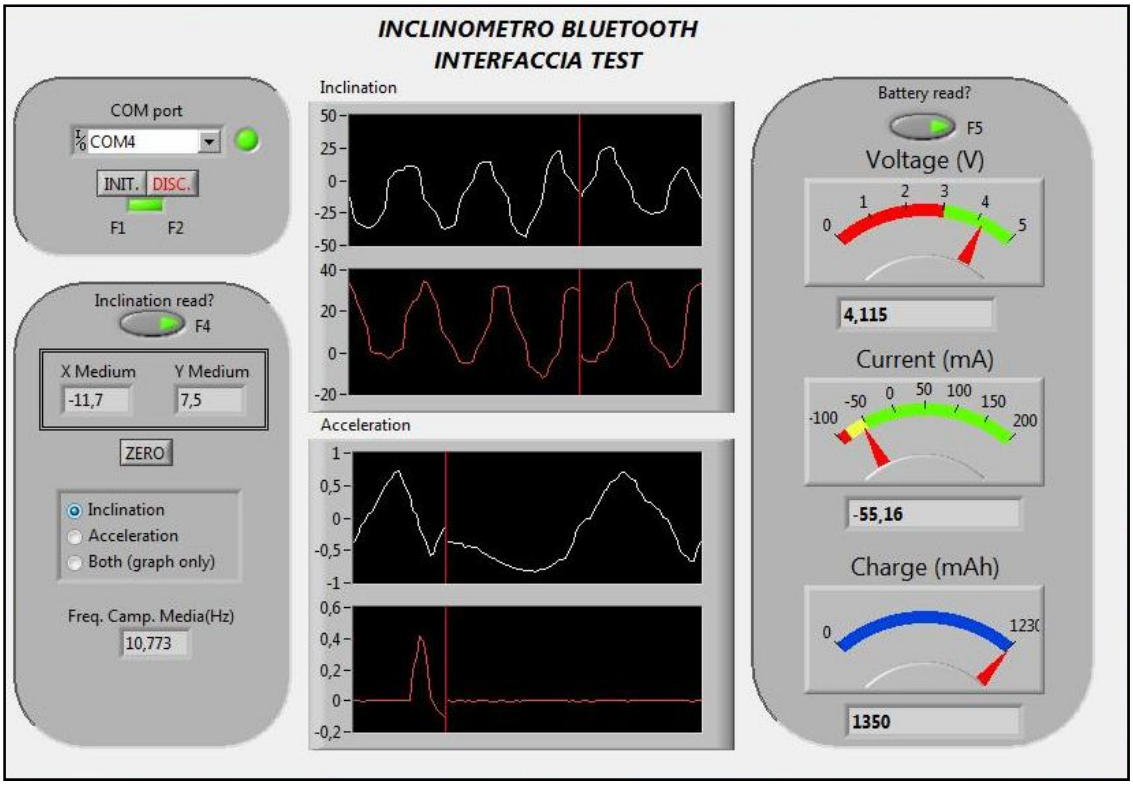


Figura 35. Inclinometer Tester

Entrambe le applicazioni sono state create utilizzando *LabVIEW* di National Instruments, e sono sostanzialmente basate su una libreria di VIs (Virtual Instruments) che raccoglie tutte le funzioni con cui è possibile interrogare l'inclinometro e decodificare i dati da esso ricevuti. I due programmi utilizzano versioni diverse della stessa libreria. *Inclinometer Tester* (Figura 35) fa riferimento alla prima versione, in cui la comunicazione era affidata ai driver NI-VISA che permettono di inviare e ricevere dati attraverso qualunque porta seriale installata sul computer. Windows installa automaticamente una porta COM per tutti quei dispositivi Bluetooth che dispongono di servizi di comunicazione seriale (SPP, vedi paragrafo 2.3). Sebbene questa soluzione offra i vantaggi di una programmazione più semplice ed universale, valida sia per la connessione Bluetooth che per quella USB (anche i chip FTDI sono riconosciuti sul computer come porte COM). Per contro, non vi è nessuna garanzia sull'associazione tra il numero di una porta COM e il dispositivo Bluetooth o il transceiver USB corrispondente. Questa associazione rimane di solito immutata nell'ambito del singolo PC, ma quando viene cambiata macchina è necessario selezionare nuovamente la porta COM corrispondente all'inclinometro Bluetooth, andando a cercarne l'identificativo nelle proprietà di sistema dello specifico PC.

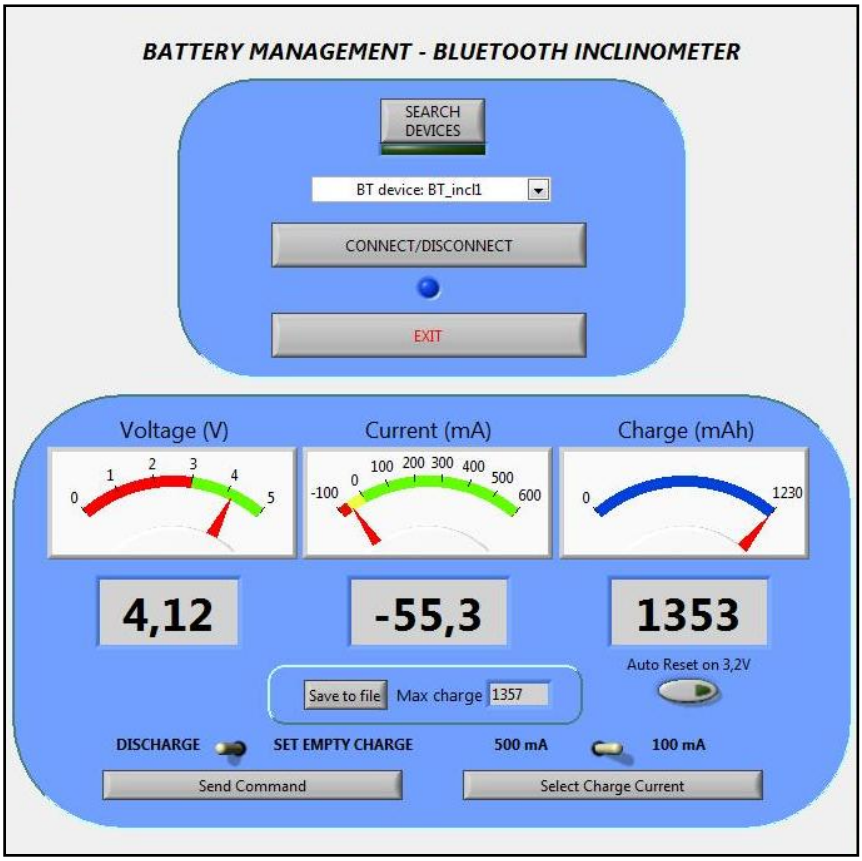


Figura 36. Inclinometer Battery Manager

La libreria utilizzata dal *Battery Manager* (Figura 36) è costruita diversamente: invece di utilizzare i driver VISA, si fa direttamente riferimento a quelli che gestiscono la radio Bluetooth presente nel PC. Per pilotare la comunicazione USB vengono usate le funzioni dei driver forniti da FTDI. In entrambi i casi il dispositivo è identificato da un indirizzo fisico univoco fissato dal costruttore dei chip e da un nome che può essere impostato all'atto della costruzione dell'inclinometro. Pertanto ogni inclinometro è identificato attraverso il nome della propria radio Bluetooth o del proprio chip FTDI (è buona norma programmare entrambi i moduli dello stesso inclinometro con lo stesso nome). Tutte le funzioni di questa libreria sono polimorfiche e compatibili con la precedente versione: è possibile selezionare, anche dinamicamente, quale tipo di driver utilizzare. Il riferimento del dispositivo passato alla VI è definito di tipo *variant* (analogo al tipo *void* definito in C/C++) e viene gestito internamente alla VI stessa per convertirlo nel tipo di riferimento adatto (porta COM, Bluetooth, USB).

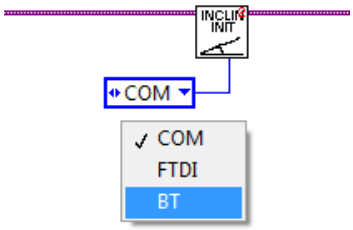


Figura 37. VI di connessione all'inclinometro

All'avvio del *Battery Manager* viene effettuata una ricerca dei dispositivi FTDI e Bluetooth installati sulla macchina e viene presentata all'utente una lista in cui sono elencati i dispositivi assieme alla tipologia del collegamento (Bluetooth o USB). Alla pressione del tasto di connessione viene selezionata l'istanza della funzione di connessione (e delle altre funzioni necessarie alla comunicazione) adatta alla scelta effettuata.

4. Misuratore di livello

4.1. Specifiche

La misura di livello deve essere effettuata con una precisione superiore a quella ottenibile con i sistemi di misura tradizionali. La sensibilità richiesta è quindi di almeno un millimetro. Come nel caso dell'inclinometro (paragrafo 3.1) è sufficiente acquisire le misure alla frequenza di 10 Hz per apprezzare le variazioni di altezza del bordo libero causate dai movimenti della superficie dell'acqua.

Il sistema deve poter essere alimentato a batteria, la quale viene ricaricata da una sorgente di alimentazione 5 V – 500 mA con connettore USB (compatibile quindi con l'inclinometro). Considerata l'entità della potenza assorbita dai sensori di posizione (oltre 2 W), si sceglie di non rendere possibile l'alimentazione dell'unità dall'esterno. La batteria deve essere in grado di erogare la potenza richiesta e deve garantire la possibilità di effettuare diverse misure nella stessa giornata senza necessità di ricarica.

4.2. Tipologia di sensore

Per effettuare la misura di livello si è scelto di utilizzare un sensore di posizione lineare magnetostrittivo (MS).

La magnetostrizione è una proprietà dei materiali ferromagnetici, quali il ferro, l'acciaio, il nickel, il cobalto, etc. Tali materiali modificano le loro dimensioni quando vengono immersi in un campo magnetico. Questo è causato dalla polarizzazione magnetica degli atomi che si dispongono in una determinata direzione. I materiali utilizzati nei sensori MS sono solitamente appartenenti ai metalli di transizione e sono trattati con processi di lavorazione a freddo e annealing termico per esaltarne la proprietà magnetostrittiva.

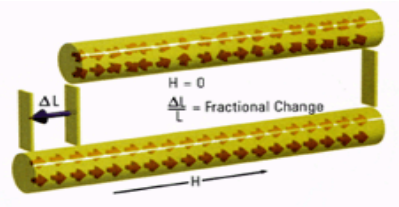


Figura 38. Effetto magnetostrittivo

Nei sensori di livello è sfruttato anche il processo inverso: applicando uno stress fisico al materiale, si registra una variazione della sua permeabilità magnetica (effetto Villari, comunemente utilizzato nei pick-up delle chitarre elettriche).

Una caratteristica importante dei fili costituiti di materiale magnetostrittivo è l'effetto Wiedemann. Quando un campo magnetico trasversale attraversa un filo percorso da corrente, si genera una torsione del filo nel punto di intersezione col campo stesso. Tale torsione è dovuta all'interazione del campo trasversale con quello indotto dalla corrente che scorre nel filo.

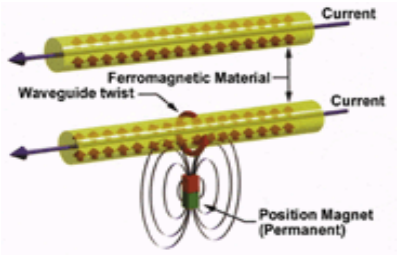


Figura 39. Effetto Wiedemann

Sfruttando la combinazione di questi effetti è possibile misurare la posizione di un magnete lungo un'asta metallica. La misura viene effettuata inviando un breve impulso di corrente (1-2 µs) lungo un filo contenuto in un rivestimento tubolare metallico, che ha la funzione di proteggere il filo e fornire robustezza meccanica al sistema. Attorno al rivestimento è situato un magnete permanente, solitamente un anello che circonda il rivestimento, libero di scorrere in entrambe le direzioni. Quando l'impulso di corrente, che si propaga lungo il filo ad una velocità prossima a quella della luce, raggiunge il campo magnetico generato dal magnete permanete, si innesca la torsione del filo per effetto Wiedemann: tale stress meccanico genera un impulso **ultrasonico** che si propaga all'interno del rivestimento, in entrambe le direzioni.

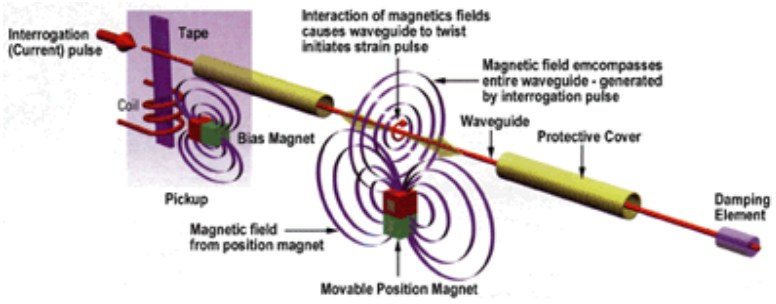


Figura 40. Misura di posizione

Mentre l'impulso che arriva alla fine del tubo viene soppresso da una apposita terminazione, quello che torna nella testa del sensore viene rilevato da una lamina metallica immersa in un campo magnetico statico. La deformazione della lamina varia la permeabilità magnetica del metallo. Da questo pick-up si ottiene un segnale in tensione che indica l'istante di ritorno dell'impulso ultrasonico. Calcolando il tempo di volo di questo impulso (il tempo di attraversamento dell'impulso di corrente è ordini di grandezza inferiore, e quindi trascurabile) è possibile risalire alla posizione del magnete lungo il filo. [12]

E' stato effettuato uno studio preliminare utilizzando un unico sensore MS dotato di uscita digitale CAN-Open, collegato ad un PC attraverso un CAN-Bus cui è connesso un transceiver NI-CAN di National Instrument. Con una semplice interfaccia *LabVIEW* è possibile configurare il sensore e ricevere i risultati delle misure effettuate. Queste vengono visualizzate su di un grafico, in tempo reale, e memorizzate su di un file.

Utilizzando questa interfaccia di base sono state effettuate alcune prove sul campo per verificare l'affidabilità e l'utilizzabilità del sistema. Visto l'esito positivo con cui si sono concluse queste prove è stato iniziato lo sviluppo di un sistema più complesso che ha come fine ultimo quello di gestire integralmente ed automaticamente una rete di sensori con cui misurare i bordi liberi di una imbarcazione. Tale sistema è descritto in dettaglio nei successivi capitoli.

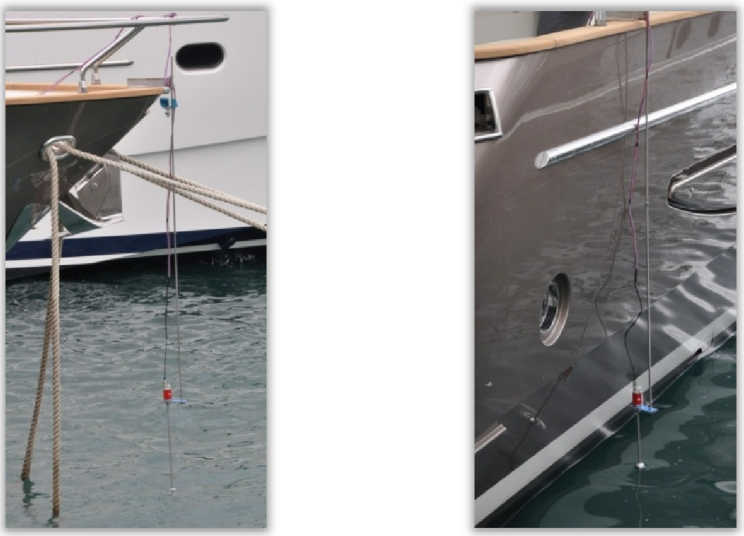


Figura 41. Misura di bordo libero effettuata con un sensore di livello magnetostrittivo (a sinistra: prua, a destra: poppa lato dritta)

Sensori di questo tipo sono prodotti da diverse aziende, sia in Europa che negli Stati Uniti. E' stata quindi svolta una ricerca per definire il prodotto più adatto, in grado di soddisfare alcuni requisiti:

- Range di misura: ≥ 500 mm;
- Misura di posizione con risoluzione: ≤ 50 µm;
- Uscita digitale facilmente interfacciabile con un microcontrollore.

Quest'ultimo punto è stato fondamentale nella scelta: diversi sensori di livello disponibili sul mercato dispongono di uscite digitali con protocolli quali CAN-Open, Profibus, Ethernet; sebbene questi siano comodi da utilizzare in ambito industriale, avrebbero bisogno di un elettronica dedicata e di un firmware piuttosto complesso per gestire lo stack di protocollo a bordo di un µC.

Figura 42. Il sensore MS prodotto da DS-Europe

La scelta è quindi ricaduta su di un prodotto dell'azienda milanese DS Europe, che dispone di uscita digitale RS485: tale protocollo è facilmente gestibile dalle risorse hardware di qualunque microcontrollore dotato di interfaccia USART, previo adattamento dei livelli elettrici. Lo standard RS485 prevede la trasmissione asincrona di dati su una linea bifilare differenziale.

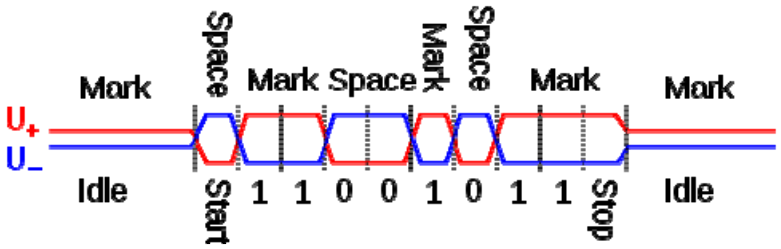


Figura 43. Esempio di trasmissione di un byte (8n1) su bus RS485

Con questo protocollo è possibile realizzare una linea multi-drop (lo standard prevede fino a 32 nodi) half-duplex (solo un nodo alla volta può trasmettere). Ogni nodo (rettangolo azzurro in Figura 44) può essere schematizzato con un trasmettitore ed un ricevitore differenziali. L'uscita del ricevitore e l'ingresso del trasmettitore sono invece riferite a massa. In figura sono evidenziate anche le abilitazioni del ricevitore (per inibire la ricezione dei dati) e del trasmettitore (per forzarne l'uscita in alta impedenza e disimpegnare la linea).

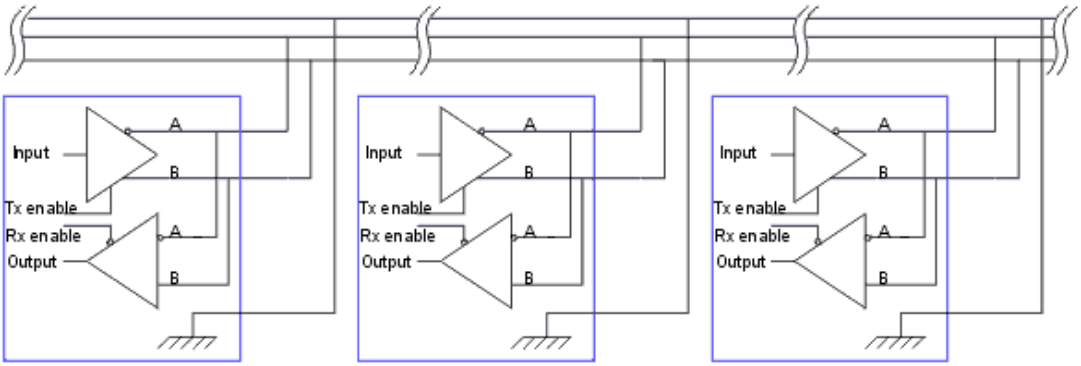


Figura 44. Linea RS485 multidrop

La scelta di questo protocollo non preclude la possibilità di realizzare una rete di misura completamente cablata: in questo caso il PC dovrà interrogare sequenzialmente i vari sensori e registrarne la risposta, assicurando che non trasmettano contemporaneamente due o più nodi. [13]

4.3. Descrizione del sistema

DESCRIZIONE DELLA MECCANICA

L'elettronica è contenuta in una scatola di plastica ABS (Hammond Manufacturing 1554G) a tenuta stagna, sufficientemente capiente per ospitare al suo interno anche la testa del sensore stesso, che contiene l'elettronica che effettua la misura del tempo di volo. La barra

cilindrica e il magnete mobile fuoriescono invece dalla scatola attraverso uno scasso praticato su una parete di essa.

Figura 45. Misuratore di livello in funzione

Per garantire l'impermeabilizzazione si ricorre a guarnizioni in gomma ed o-ring in corrispondenza della battuta tra la testa del sensore e la parete interna della scatola (vedi paragrafo 4.3). All'interno della scatola il sensore è connesso alla scheda con la restante elettronica mediante un connettore standard M12. Le operazioni di accensione e spegnimento dell'unità sono controllate mediante un pulsante (ITW Switches 57-111) posto

sulla scatola stessa. L'utente è informato sullo stato del dispositivo, oltre che sull'unità centrale, direttamente sulla singola unità di misura, mediante segnalazioni luminose a LED. Per la ricarica della batteria si usa un connettore di tipo mini-USB (Bulgin PX0443) al quale collegare una sorgente di alimentazione 5 V – 500 mA. Tutte le interfacce che mettono in comunicazione l'elettronica con l'esterno della scatola devono garantire una grado di protezione da polvere ed umidità uguale o superiore a quello della scatola stessa (IP66). Per quanto riguarda il pulsante ed i connettori ci si è affidati a componenti certificati dal costruttore, che realizza l'impermeabilizzazione mediante guarnizioni in gomma. Le segnalazioni luminose sono invece effettuate con LED cilindrici da 5 mm: questi sono stati montati in un foro di diametro opportuno ed allineati a raso con la superficie esterna della scatola). Quindi è stato deposto uno strato di adesivo termoplastico dall'interno della scatola. Questo sistema garantisce una buona tenuta finché la pressione esterna non raggiunge valori tali da distaccare il nylon dalla superficie interna della scatola. In una successiva fase di ingegnerizzazione si suggerisce di sostituire tali LED con specifici indicatori da pannello con grado di protezione adeguato e certificato dal costruttore.

Per rendere possibile l'ancoraggio del sistema e il collegamento ad aste metriche, utili per posizionarlo con precisione, la scatola verrà corredata con un collare in acciaio che fornisce rigidità e robustezza, munito di giunto filettato alla sommità, sul quale è possibile il collegamento di aste di prolunga.

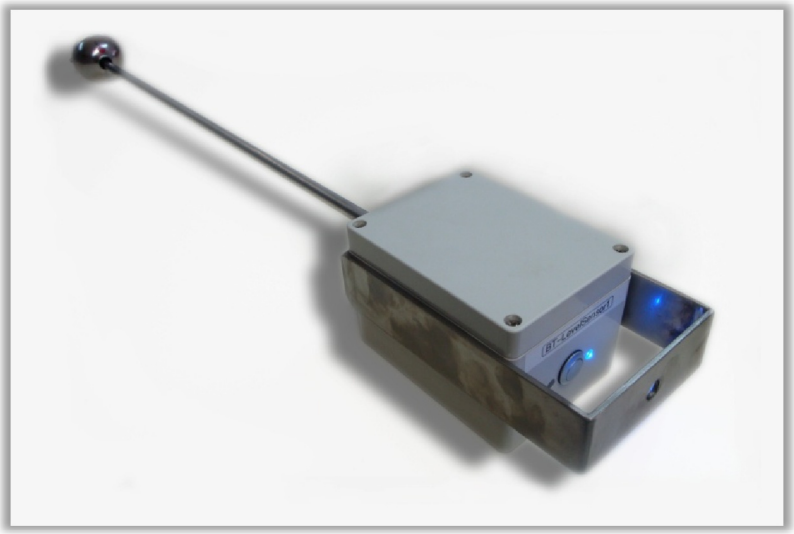


Figura 46. Misuratore di livello con staffa di ancoraggio montata

DESCRIZIONE DELL'ELETTRONICA

Nello schema seguente sono illustrati i blocchi costitutivi della scheda elettronica e sono indicati i nomi dei circuiti integrati utilizzati.

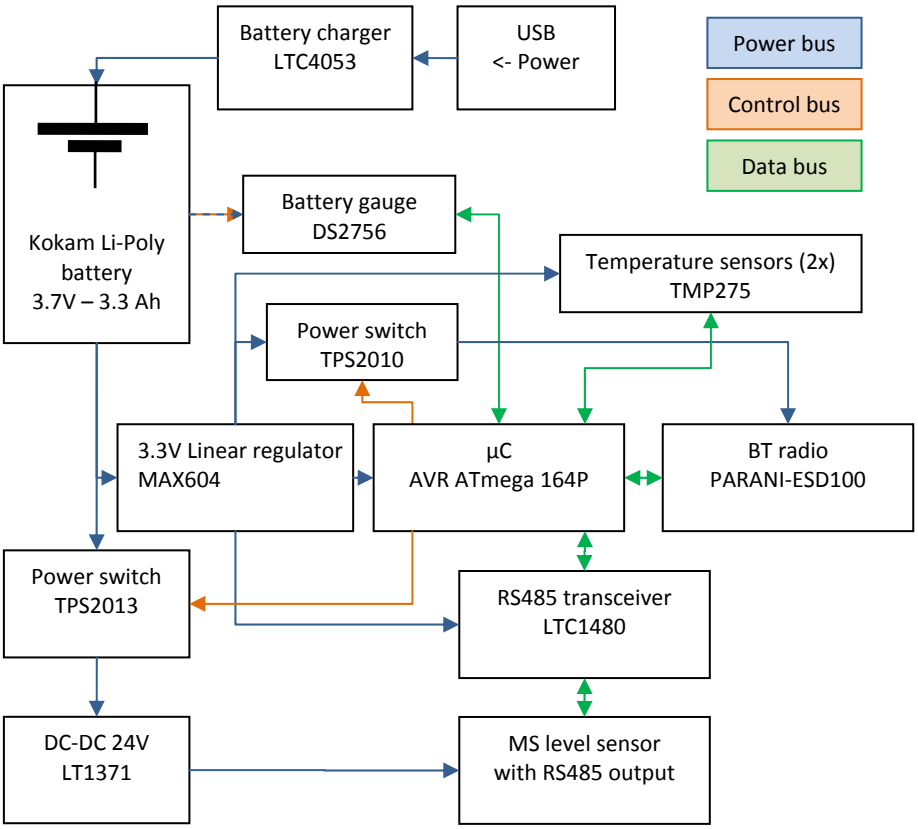


Figura 47. Schema generale del misuratore di livello

I quattro elementi che scambiano dati tra di loro sono:

- sensore di livello;
- modulo bluetooth Parani ESD-100;
- misuratore di carica della batteria;
- sensori di temperatura.

Questi sono collegati al microcontrollore AVR, sfruttando protocolli di comunicazione diversi. Il sensore MS utilizza la comunicazione seriale differenziale RS485, il modulo BT comunica con protocollo seriale asincrono con livelli CMOS, il misuratore di carica invece è interfacciato con un collegamento one-wire (vedi par. 3.3), i sensori di temperatura sono entrambi collegati ad un SM-Bus I^2C.

Nella pagina successiva è riportato lo schema elettrico del dispositivo.

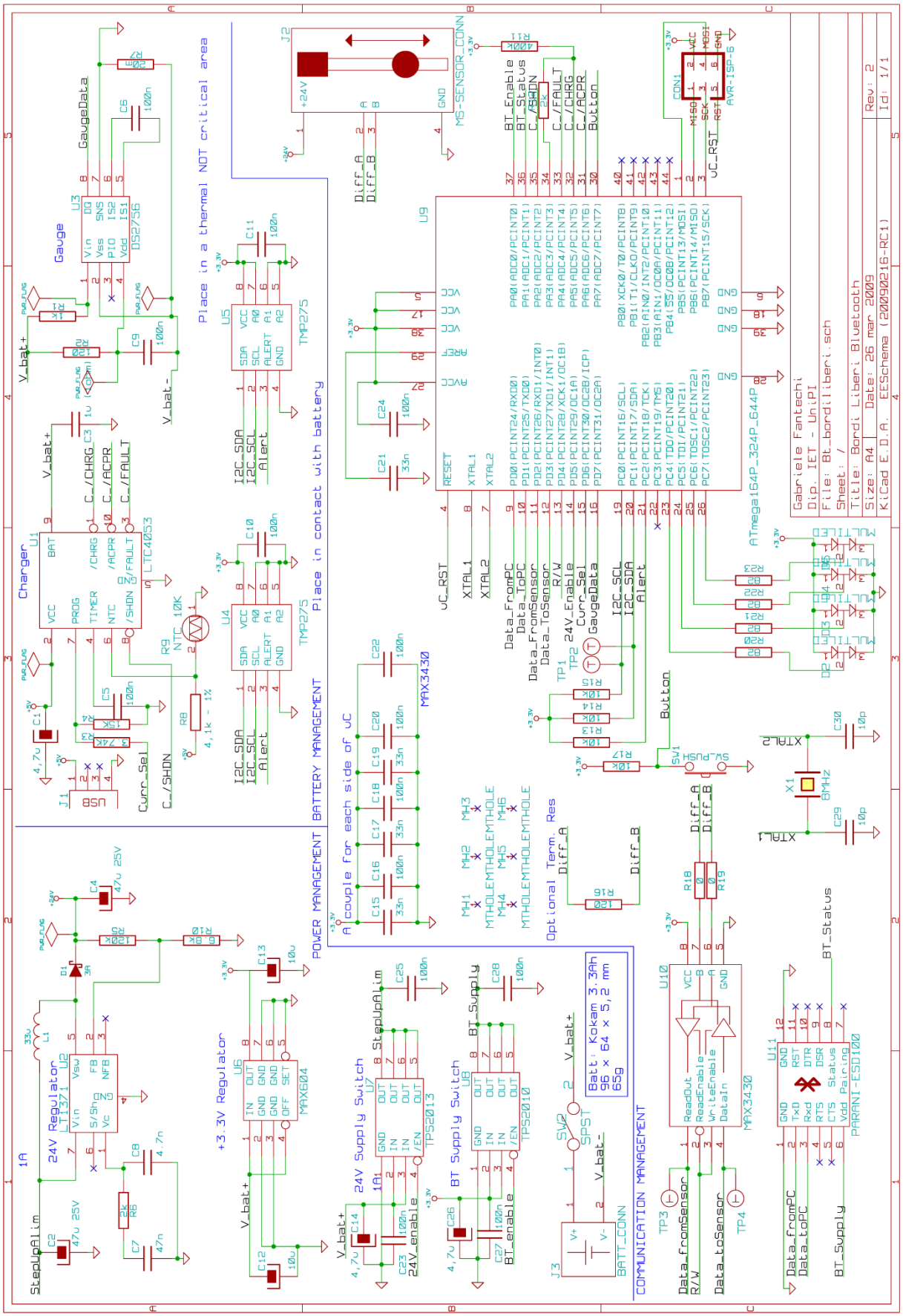

Place in a thermal NOT critical area
Place in contact with battery
POWER MANAGEMENT
BATTERY MANAGEMENT
COMMUNICATION MANAGEMENT
Gauge
Charger
24V Regulator
+3.3V Regulator
24V Supply Switch
BT Supply Switch
ATmega164P_324P_644P
MAX3430
PARANI-ESD100
LTC4053
DS2756
TMP275
MAX4430
TPS2013
TPS2010
MAXE04
LT1371
MS-SENSOR_CONN
AVR-ISP-6
Gabriele Fantechi
Dip. IET - UniPI
File: Bt_bordiliberi.sch
Sheet: /
Title: Bordi Liberi Bluetooth
Size: A4
Date: 26 mar 2009
Rev: 2
KiCad E.D.A. EESchema (20090216-RC1)
Id: 1/1
Batt: Kokam 3.3Ah
96 x 64 x 5,2 mm
65g
GaugeData
V_bat+
V_bat-
I2C_SDA
I2C_SCL
Alert
C_/CHRG
C_/ACPR
C_/FAULT
C_/SHDN
Curr_Sel
Button
BT_Enable
BT_Status
uC_RST
XTAL1
XTAL2
Data_FromPC
Data_ToPC
Data_FromSensor
Data_ToSensor
R/W
24V_Enable
StepUpAlim
BT_Supply
Diff_A
Diff_B
Data_fromPC
Data_toPC
Data_fromSensor
Data_toSensor

Figura 48. Schema elettrico del misuratore di livello

Il microcontrollore gestisce le richieste ricevute dall'unità centrale mediante collegamento wireless, acquisisce dati (dal sensore di livello, dal misuratore di carica della batteria, dai sensori di temperatura) e li ritrasmette all'unità stessa.

Per la comunicazione Bluetooth è stato scelto il modulo seriale-Bluetooth Sena Parani ESD-100, Classe 1, usato anche per l'inclinometro. Questo è interfacciato al µC mediante le connessioni seriali RXD e TXD e di status (attivo basso).

Il dispositivo è normalmente spento (non si ha una comunicazione attivata e sia il trasmettitore che il sensore di livello risultano spenti). Alla pressione del tasto si attiva l'accensione della radio Bluetooth (U11) comandata dal microcontrollore. L'operazione sarà segnalata da un opportuno lampeggiamento di un LED blu. L'attivazione della radio avviene generando un livello logico basso sull'ingresso di */ENABLE* di U8 (switch high-side TPS2010) che rende attivo il collegamento della VCC 3.3 V con l'alimentazione di U11. Questo chip è collegato adottando la configurazione circuitale consigliata dal costruttore. [10]

L'avvenuta connessione radio con un dispositivo nelle vicinanze è segnalata da opportuno lampeggiamento del LED, e il µC riconosce questo stato campionando l'uscita STATUS di U11.

L'alimentazione per tutti i chip è fornita dal regolatore MAX604 (U6) che regola a 3,3 V la tensione d'ingresso da batteria.

Figura 49. Batteria Kokam ai polimeri di litio

Con alimentazione 5V-500mA collegata si ricarica la batteria ai polimeri di litio (3,7 V - 3300 mAh, prodotta da Kokam) con il chip LTC4053 (U1). Durante la carica della batteria non è sarà permesso di alimentare il sensore di livello: in tale condizione si avrebbe un assorbimento di corrente maggiore di quello fornito dall'alimentatore esterno, pertanto la batteria si scaricherebbe anziché caricarsi. Il caricabatteria U1 sarebbe così forzato in condizione di errore.

Ripetendo la scelta fatta per l'inclinometro, si prevede di poter caricare la batteria con 2 diversi possibili valori di corrente, 100 mA o 500 mA, selezionabili dall'interfaccia di controllo su PC: la configurazione circuitale del chip è la stessa descritta nel paragrafo 3.3.

In questo caso però si è scelto di realizzare un controllo di temperatura della batteria direttamente a bordo del caricabatterie, collegando esternamente un termistore NTC (R9). Qualora la temperatura della batteria superi la soglia di 50 °C o scenda sotto 0 °C la carica viene automaticamente interrotta. Questa informazione può essere acquisita dal microcontrollore attraverso i pin di */FAULT* e */CHRG* del caricabatterie.

Dal momento che viene utilizzata una batteria ad alta densità di energia è utile monitorarne la temperatura da PC, onde evitarne il danneggiamento[12]. Per ottenere tale informazione si utilizzano due sensori digitali di temperatura TMP275. Questi sono interfacciati con il µC attraverso un **SMBus I²C**.

Si tratta di un protocollo di comunicazione seriale sincrono, che utilizza due linee: *SDA* (Serial Data line) e *SCL* (Serial Clock line).

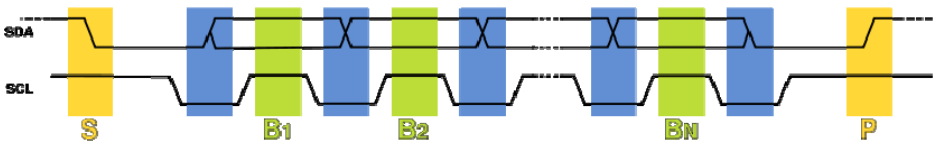


Figura 50. Segnali della comunicazione I2C

Tutti i dispositivi del bus sono connessi a queste linee per mezzo di porte *open-drain*, è pertanto necessario un pull-up resistivo per ogni linea (per realizzare una wired-AND).

[12] La ricarica errata di batterie di questo tipo può provocarne l'incendio o l'esplosione. Anche uno stress meccanico che porti al danneggiamento dell'involucro esterno può provocare l'ingresso di umidità all'interno e conseguentemente una reazione che sviluppa gas gonfiando l'involucro stesso.

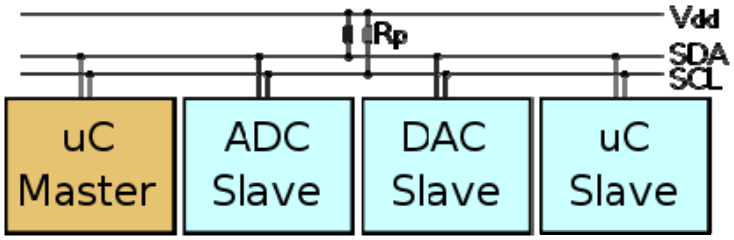


Figura 51. Connessione dei dispositivi all'SMBus

La linea di clock è pilotata esclusivamente dal dispositivo *master*, mentre la linea dati può essere controllata alternativamente da *master* o *slave*.

I dispositivi collegati al bus sono identificati da un indirizzo a 7 bit, ma 16 indirizzi sono riservati. E' quindi possibile connettere fino a 112 dispositivi allo stesso bus.

Nel caso dei TMP275 vengono trasmessi sempre 2 byte: all'inizio di ogni comunicazione il dispositivo *master* (µC) dichiara l'indirizzo di un dispositivo slave nel primo byte e l'indirizzo del registro da leggere nel secondo byte. Quindi il sensore risponde con 2 byte contenenti i dati richiesti dal *master*. L'indirizzo di ogni sensore può essere selezionato collegando opportunamente i 3 pin *A[2..0]* dei TMP275 (possono essere collegati 8 sensori sullo stesso bus). Un sensore è posto in contatto termico con la batteria, l'altro è posizionato sull'altro lato del PCB: in questo modo è possibile ricavare una misura molto precisa (ogni TMP275 fornisce un'accuratezza di 0,5 °C) dell'incremento di temperatura della batteria rispetto all'ambiente circostante.

Con il battery gauge DS2751 (U3) si effettua il controllo dello stato di carica della batteria. L'informazione viene trasmessa al µC attraverso interfaccia 1-Wire, descritta nel paragrafo 3.3. Tali informazioni sono quindi trasferite all'interfaccia di controllo.

Il sensore di livello deve essere alimentato a 24 V ed assorbe al massimo 100 mA. Si ottiene tale alimentazione attraverso un convertitore DC-DC switching, realizzato con l'integrato LT1371 (U2), montato in configurazione boost.

Per il dimensionamento dei componenti passivi sono stati svolti dei semplici calcoli approssimando al primo ordine tutte le variazioni di corrente e tensione, considerando costante la corrente assorbita dal carico, la tensione di ingresso e quella di uscita e

ponendosi in condizione di regime. I risultati dei calcoli sono stati successivamente integrati con i suggerimenti del costruttore. [14]

$$\delta = \frac{V_{out} - V_{in}}{V_{out}} = \frac{24 - 3{,}5}{24} \cong 0{,}85; t_{sc} = T_{sw} \cdot \delta = \frac{\delta}{f_{sw}} = \frac{0{,}85}{5 \cdot 10^5} = 1{,}7\ \mu s;\ \Delta I_c = \frac{V_{in} \cdot\ t_{sc}}{L};$$

$$L_{min} = \frac{V_{in_{max}} \cdot\ t_{sc}}{\Delta I_{c_{max}}} = \frac{4{,}2 \cdot 1{,}7 \cdot 10^{-6}}{0{,}5} \cong 15\ \mu H$$

Fissato il valore massimo accettabile del ripple di corrente a 0,5 A (funzionamento in Continuos Current Mode), è necessario utilizzare un'induttore di almeno 15 µH. Il valore commerciale scelto è di 33 µH.

Quando lo switch è chiuso (quindi per 1,7 µs), la corrente è fornita al carico esclusivamente dalla capacità presente sull'uscita.

$$\Delta V = \frac{I}{C};\ C = \frac{I \cdot t_{sc}}{\Delta V} = \frac{0{,}1 \cdot 1{,}7 \cdot 10^{-6}}{0{,}1} = 1{,}7\ \mu F$$

Per ottenere un ripple di tensione di ampiezza inferiore a 0,1 V si userà un condensatore di 1,7 µF. Anche in questo caso è stata effettuata una scelta cautelativa fissandone il valore a 47 µF.

Nel firmware è implementata un'efficiente gestione dell'energia, permettendo al sistema di operare in condizioni di normale utilizzo per oltre 15 ore di utilizzo tipico. Il risparmio energetico è ottenuto alimentando il sensore di livello (l'elemento del sistema che assorbe la maggior parte della potenza) solo per il tempo strettamente necessario ad effettuare la misura.

L'unità centrale può connettersi al sistema di misura tramite rete bluetooth in qualunque momento, anche durante il posizionamento dei tre sensori sullo scafo dell'imbarcazione. Questo permette l'identificazione dei tre sensori e il corretto posizionamento. Al momento in cui l'utente scende dall'imbarcazione e dà il via alle operazioni di misura, al microcontrollore viene inviato il comando per alimentare il sensore. Una volta effettuata la misura, che può durare qualche decina di secondi dipendentemente dalle condizioni del

mare, il sensore viene di nuovo disalimentato. La disattivazione del convertitore DC-DC è effettuata inserendo a un secondo switch high-side, analogo a U8, salvo per il valore massimo della corrente che può sopportare (si usa infatti un TPS2013, U7).

Il transceiver TTL-RS485 LTC1480 (U10) realizza l'adattamento dei livelli tra il sensore di livello ed il microcontrollore. [15]

Oltre alle capacità di filtraggio suggerite dai costruttori dei componenti, sono stati inseriti condensatori di bypass da 100 nF e/o 33 nF sul alimentazione, in prossimità dei vari circuiti integrati.

Effettuando alcune semplici ipotesi è possibile stimare il tempo di durata della batteria.

Il sensore assorbe al massimo 100 mA a 24 V ovvero circa 700 mA a 3.5 V considerando un efficienza di conversione dell'85 %. Il resto dell'elettronica sulla scheda assorbe circa 60 mA. Il sistema è alimentato con una batteria ai polimeri di litio ad alta capacità (3.3Ah): questa garantisce un'autonomia di circa 4 ore a pieno carico. Per la ricarica, effettuata mediante alimentatore DC 5 V impiega circa 6 ore se viene selezionata la corrente di ricarica di 500 mA mentre saranno necessarie oltre 36 ore utilizzando 100 mA.

Durante il normale utilizzo del sistema di misura possono essere necessari 30 minuti complessivi per il montaggio e lo smontaggio del sistema. Se ipotizziamo di mantenere abilitata la radio Bluetooth per tutto il tempo, si consumeranno 30 mAh. Effettuando la misura per 60 secondi si utilizzano altri 13 mAh.

Si è scelto di sovrastimare il consumo totale per una misura completa con 50 mAh. E' quindi evidente che con tutte le batterie completamente cariche si possono effettuare oltre 60 misure.

Per le segnalazioni luminose sono utilizzati due LED, uno di colore blu (TT Electronics), che identifica lo stato della connessione Bluetooth, mentre il secondo è un LED bicolore rosso/verde (Kingbright) che fornisce informazioni sullo stato della batteria. Le segnalazioni che vengono emesse sono elencate nella tabella seguente.

Stato del sistema	LED Bluetooth	LED Alimentazione
Spento	Spento	Spento
Bluetooth in attesa di connessione	Blu fisso	Spento
Bluetooth connesso	Blu lampeggiante	Spento
Bluetooth connesso, misura in corso	Blu lampeggiante	Rosso/Verde alternato*
Batteria quasi scarica	[dipendente BT]	Rosso lamp. lento
Batteria scarica	[dipendente BT]	Rosso lamp. veloce
Alimentazione esterna presente, batteria in carica	[dipendente BT]	Verde lampeggiante
Alimentazione esterna presente, carica completata	[dipendente BT]	Verde fisso
Alimentazione esterna presente, errore nella carica	[dipendente BT]	Rosso fisso

** tale segnalazione ha la priorità su ogni altra segnalazione relativa alla batteria.*

Tabella 5. Segnalazioni luminose del misuratore di livello

DESCRIZIONE DEL FIRMWARE

Sono implementate nel microcontrollore tre macchine a stati distinte: una dedicata alla gestione della connessione Bluetooth, una si occupa della batteria e della sua ricarica, una terza della comunicazione. I grafi successivi illustrano la logica utilizzata dalle prime due, la terza è descritta in seguito.

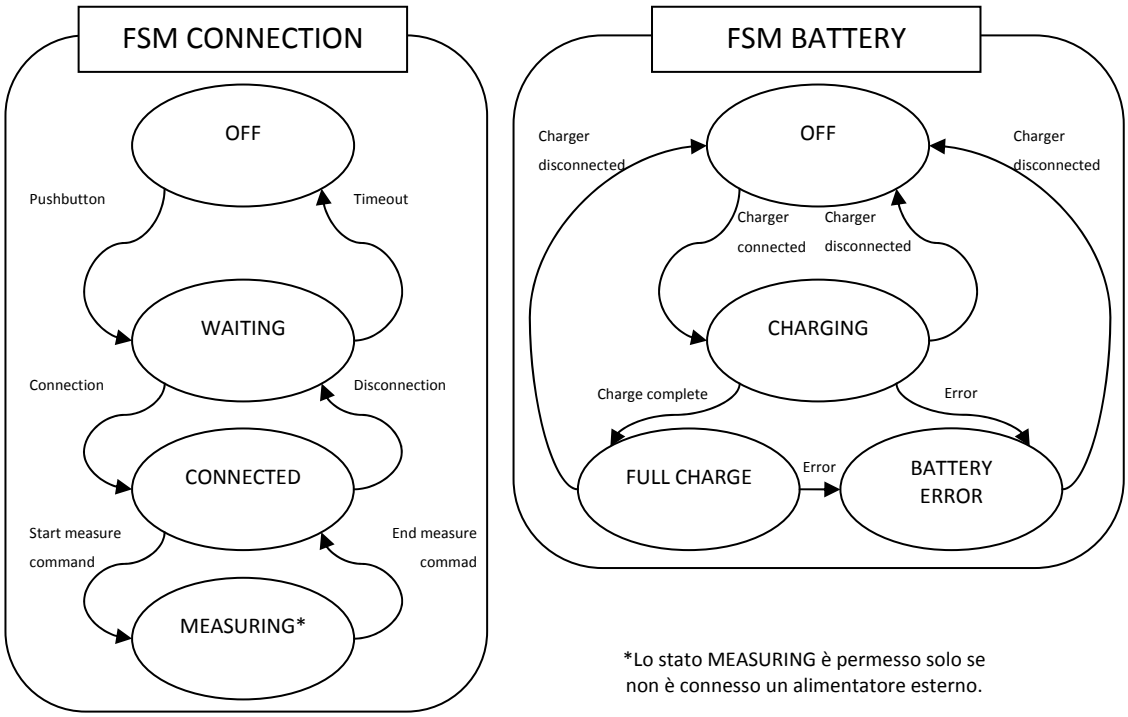


Figura 52. Macchine a stati (misuratore di livello) di gestione della connessione e della batteria

La struttura del firmware è analoga a quella inclinometro, descritta nel paragrafo 3.3. Il programma del µC viene eseguito ad una velocità di 8 milioni di operazioni al secondo: è

stato però inserito un meccanismo di sincronizzazione sfruttando le strutture hardware dedicate alla realizzazione di timer e gli interrupt da esse generate. Con questo accorgimento le macchine a stati sono aggiornate ogni 10 ms. Quando la macchina FSM CONNECTION si trova nello stato CONNECTED è possibile inviare comandi al µC via Bluetooth. Sarà quindi possibile accedere alle informazioni di batteria nonché attivare l'alimentazione del sensore di livello. Solo e soltanto quando ciò è avvenuto si passa nello stato MEASURING ed è quindi possibile comunicare attivamente col sensore. Occorre precisare che la radio BT e il sensore di livello, seppur comunicando con stringhe asincrone, non necessariamente li invieranno alla stessa velocità. In particolare, il sensore di livello è programmato dal costruttore per trasmettere stringhe 8n1 a 57600 bps mentre la radio Bluetooth viene configurata 8n1 a 115200 bps. Appare evidente che il microcontrollore deve anche fornire una funzione di bufferizzazione per poter lavorare con due baud-rate differenti. Si è scelto di limitare la dimensione dei buffer a 16 byte, sufficienti ad ospitare tutti i comandi utilizzati e le relative risposte. Inoltre limitando la dimensione dei messaggi ci si assicura che le varie trasmissioni avvengano all'interno dello slot temporale di 10 ms.

I comandi riconosciuti dall'unità di misura si suddividono in due categorie:

- **comandi da inoltrare al sensore di livello**

  I comandi appartenenti al questo gruppo iniziano con il carattere "@" (at) e devono rispettare le regole fornite dal costruttore del sensore [16] (vedi Appendice D). Sebbene sia possibile inviare al sensore qualunque comando, durante l'utilizzo normale saranno utilizzati dall'applicativo *LabVIEW* soltanto i comandi di taratura ("L" e "T"), di lettura ("R"). Ogni comando inviato al sensore deve includere un numero che identifica univocamente il sensore stesso. Questo permette la risoluzione dei conflitti in un bus RS-485 condiviso, ad opera di un unità master. Nel sistema qui descritto una sola unità è connessa al bus, e l'identificazione delle diverse unità di misura è basata sull'indirizzo fisico (MAC) delle radio bluetooth. Per questo motivo i comandi inviati dall'applicativo *LabVIEW* conterranno il carattere jolly "?" che bypassa il controllo sull'ID dei sensori e interroga simultaneamente tutti i sensori fisicamente collegati allo stesso bus. In una implementazione del sistema di misura basata su rete RS-485 filata sarà necessario

tenere conto di questo aspetto. Al comando "R" seguirà una risposta da parte del sensore, nel formato *Numero cursore + R + posizione*: quest'ultima è espressa da 7 cifre che esprimono la posizione direttamente in unità metriche (previa taratura). I comandi di taratura invece servono per assegnare a due posizioni del galleggiante le etichette di *ZERO* e *FONDOSCALA* (comando "T") alle quali sono assegnati tramite il comando "L" i valori numerici corrispondenti. In seguito ad una taratura è necessario disalimentare e rialimentare nuovamente il sensore MS per permettere il ricalcolo dei coefficienti interni. Lo schema in Figura 53 illustra le operazioni effettuate ad ogni comunicazione col sensore e le strategie di gestione degli errori di timeout (ovvero quando non viene inviato il carattere di fine comando <cr> entro un certo tempo, oppure i casi in cui non si riceve risposta dal sensore di livello). Tale algoritmo è implementato in una terza FSM.

- **comandi di servizio e di gestione della batteria**

  Al secondo gruppo appartengono tutti i comandi necessari al controllo delle operazioni che il microcontrollore deve eseguire a bordo della scheda. Quest'ultimo è programmato in modo da riconoscere determinate stringhe, effettuare le operazioni necessarie ed inviare una risposta verso il PC. Nel caso venga ricevuto un comando di lettura dello stato della batteria viene inviato il dato memorizzato nella memoria del microcontrollore: questo viene aggiornato periodicamente interrogando il battery fuel-gauge tramite connessione one-wire. Occorre tenere presente che la comunicazione one-wire è gestita completamente utilizzando ritardi e temporizzazioni sofware. Per questo motivo eventuali routine di interrupt che vadano a cadere durante una trasmissione provocherebbero errori di temporizzazione. Non è possibile mascherare le interruzioni, nemmeno per un breve periodo, poiché queste gestiscono anche la ricezione delle periferiche UART. Si ricorre quindi ad un flag che viene settato ogni volta che viene eseguita una ISR: quando si inizia la comunicazione one-wire il flag viene posto a 0; se alla fine della comunicazione il flag viene trovato a 1, il dato letto viene scartato. Quando vengono inviati comandi per attivare o disattivare l'alimentazione del sensore di livello, sono settati gli opportuni livelli di tensione sulla linea che pilota lo switch hi-side. Altri comandi sono dedicati alla lettura delle temperature, al controllo del caricabatteria e alla relativa gestione degli errori.

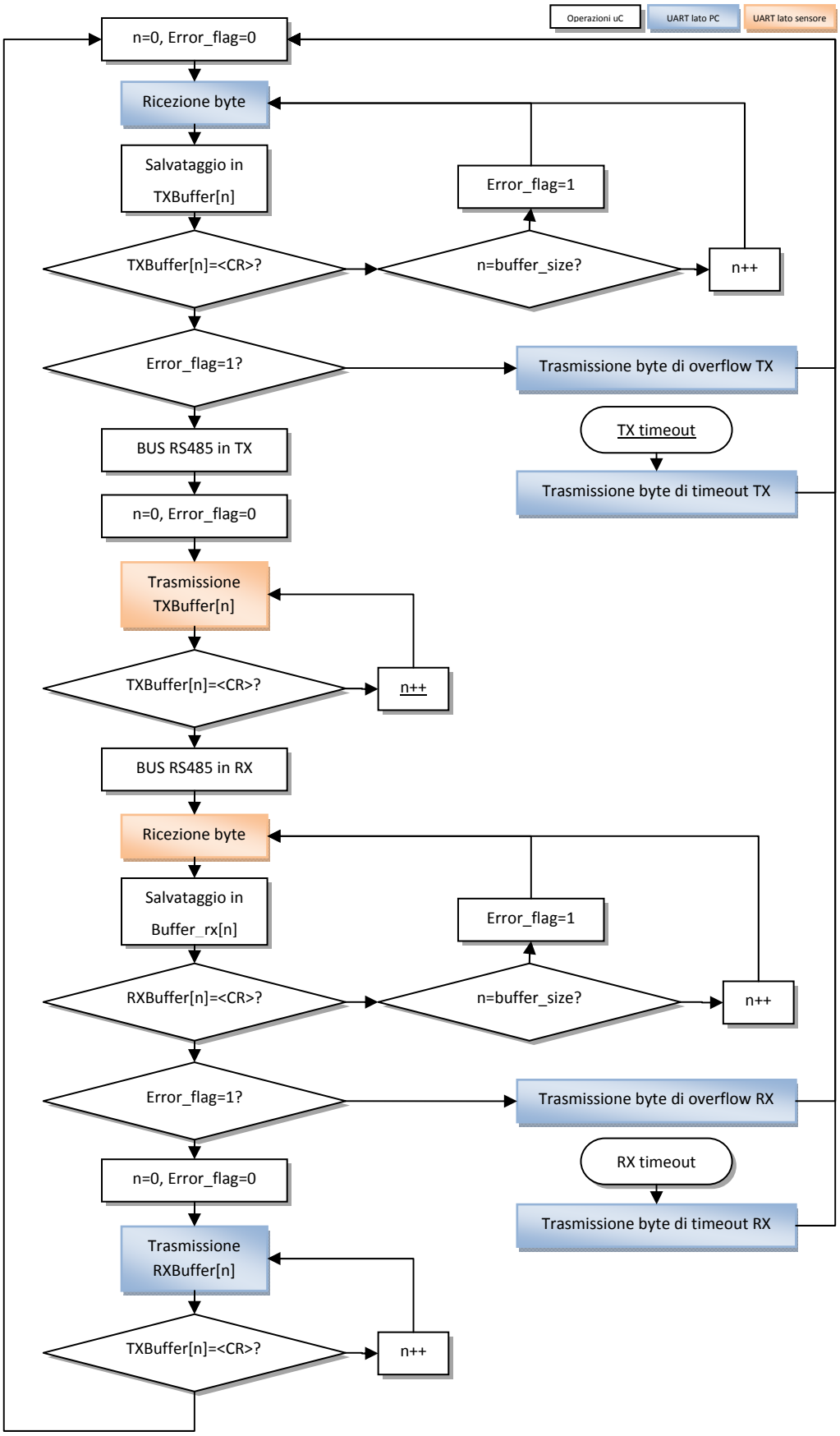

Operazioni uC
UART lato PC
UART lato sensore
n=0, Error_flag=0
Ricezione byte
Salvataggio in TXBuffer[n]
Error_flag=1
TXBuffer[n]=<CR>?
n=buffer_size?
n++
Error_flag=1?
Trasmissione byte di overflow TX
TX timeout
Trasmissione byte di timeout TX
BUS RS485 in TX
n=0, Error_flag=0
Trasmissione TXBuffer[n]
TXBuffer[n]=<CR>?
n++
BUS RS485 in RX
Ricezione byte
Salvataggio in Buffer_rx[n]
Error_flag=1
RXBuffer[n]=<CR>?
n=buffer_size?
n++
Error_flag=1?
Trasmissione byte di overflow RX
RX timeout
Trasmissione byte di timeout RX
n=0, Error_flag=0
Trasmissione RXBuffer[n]
TXBuffer[n]=<CR>?
n++

Figura 53. Elaborazione dei comandi

In Tabella 6 sono elencati tutti i possibili comandi che possono essere gestiti dal microcontrollore, in Tabella 7 i possibili messaggi di errori che possono essere trasmessi dall'unità.

Comando	Operazione	Risposta e decodifica
@ … <cr>	Comando indirizzato al sensore MS	Dipendente dal comando inviato (vedi Appendice D)
V24ON<cr>	Accensione del sensore di livello	!<cr>
V24OFF<cr>	Spegnimento del sensore di livello	!<cr>
V<cr>	Lettura tensione batteria	0V[2 Byte: S+10 bit Left Justified – 4,88mV/bit] <cr>
I<cr>	Lettura corrente batteria	0I[2 Byte: S+12 bit Left Justified – 0,78125 mA/bit] <cr>
C<cr>	Lettura carica batteria	0C[2 Byte: S+15 bit Left Justified – 312,5 µAh/bit] <cr>
K<cr>	Reset valore di carica	!<cr>
TB<cr>	Lettura temperatura della batteria	0T[2 Byte: 2's comp. 12 bit Left Justified]<cr>
TA<cr>	Lettura temperatura ambiente	0T[2 Byte: 2's comp. 12 bit Left Justified]<cr>
C_ON<cr>	Abilitazione ricarica della batteria	!<cr>
C_OFF<cr>	Disabilitazione ricarica della batteria	!<cr>
H<cr>	Selezione corrente di ricarica: 500mA	!<cr>
L<cr>	Selezione corrente di ricarica: 100mA	!<cr>
VER<cr>	Lettura versione del firmware	V[n.versione]<cr>
Altro	Comando non gestito	UNKNOWN<cr>

Tabella 6. Comandi del misuratore di livello

Messaggio	Errore
E1<cr>	Timeout nella ricezione del comando dal PC
E2<cr>	Eccessiva lunghezza del comando ricevuto dal PC
E3<cr>	Timeout nella ricezione del comando dal sensore
E4<cr>	Eccessiva lunghezza del comando ricevuto dal sensore
E5<cr>	E' stato inviato un comando al sensore quando questo non è alimentato
E6<cr>	Impossibile alimentare il sensore (batteria in ricarica o troppo scarica)

Tabella 7. Messaggi di errore del misuratore di livello

4.4. Realizzazione del PCB e collaudo

Durante la fase del progetto geometrico del PCB di questo sistema sono state affrontate alcune problematiche legate principalmente alla gestione della potenza.

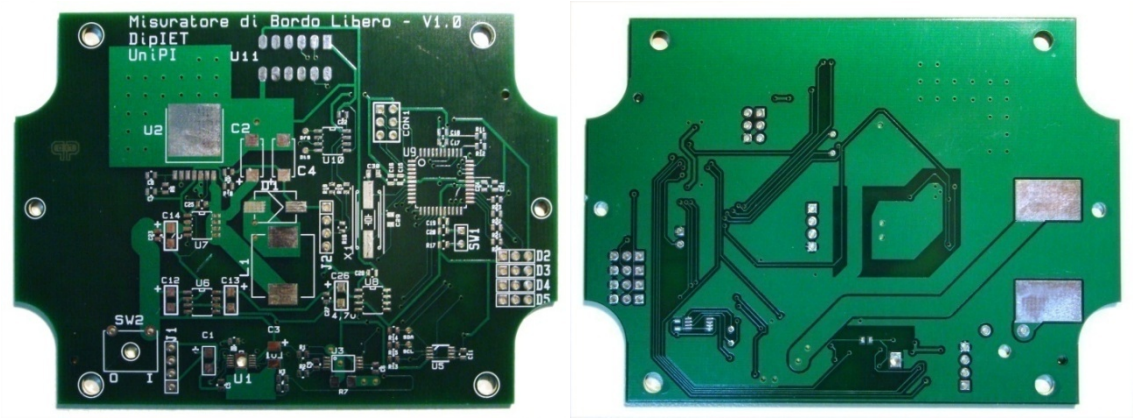


Figura 54: PCB del sensore di livello

Il circuito integrato che realizza l'alimentatore switching che fornisce l'alimentazione a 24 V è relativamente delicato: si tratta di un componente a montaggio superficiale, con package derivato dal classico through-hole TO-220. Il costruttore dichiara un'efficienza di conversione di circa 85 %: dovendo convertire una potenza di circa 2 W, il chip è stato dotato di un adeguato dissipatore di calore. Disponendo di sufficiente area sulla scheda si è scelto di realizzare questo dissipatore mediante un'area di rame di dimensioni adeguate (si ottiene una resistenza termica θ_{JA}<65 °C/W, così da garantire il funzionamento del dispositivo fino ad una temperatura ambiente di 60 °C). L'area di rame realizzata ha una superficie di oltre mezzo pollice quadrato, ed è collegata al piano di massa presente sull'altro lato della scheda mediate una matrice di via.

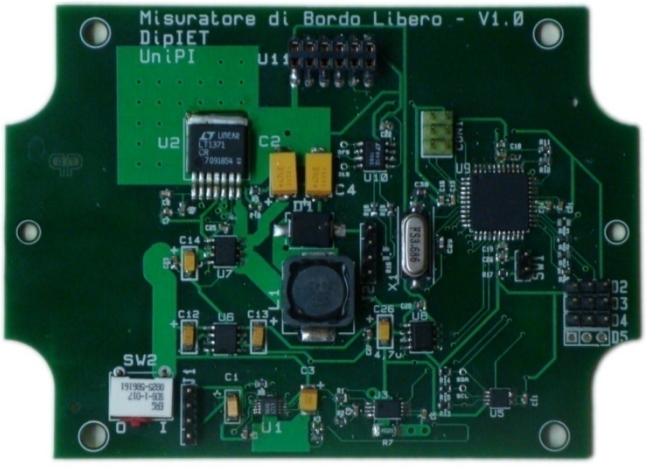


Figura 55. PCB del misuratore di livello con componenti saldati

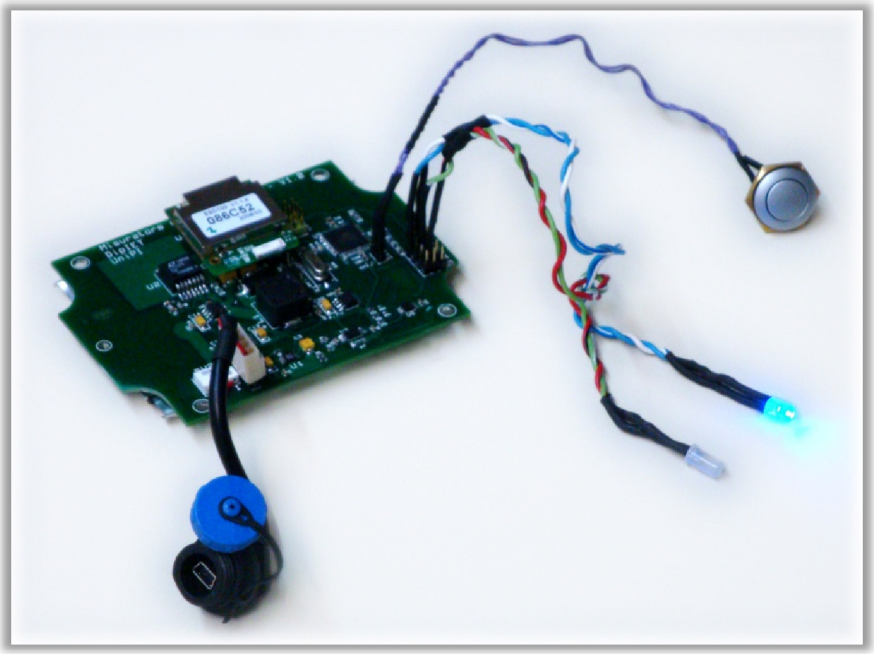


Figura 56. Scheda del misuratore di livello, montata

Per ridurre l'effetto delle interferenze elettromagnetiche generate dal convertitore switching, che lavora alla frequenza fissata di 500 KHz, è stata minimizzata la dimensione delle piste che collegano il chip con l'induttore (L1), il diodo di ricircolo (D1) e il condensatore di livellamento (C4). Inoltre si è scelto di utilizzare un induttore avvolto su nucleo di ferrite e completamente schermato. In Figura 57 sono evidenziate le suddette geometrie: in blu la net GND, in arancio l'alimentazione proveniente dalla batteria, in verde (chiaro sul lato rame) il nodo in commutazione, in marrone l'uscita a 24 V.

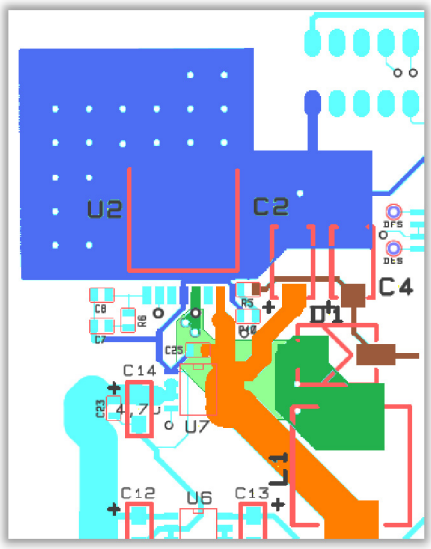


Figura 57. Dettaglio del layout del convertitore DC/DC

La batteria (in rosso in Figura 58) è connessa direttamente al PCB (in verde) tramite due linguette saldate su di esso (in azzurro). È alloggiata sul fondo della scatola (in nero) e rimane tra il fondo della scatola ed il PCB. Tra la batteria e il PCB è interposto un foglio di plastica semi-rigida per evitare che i reofori dei componenti through-hole danneggino la superficie della batteria. Tale lamina presenta due aperture in corrispondenza del sensore di temperature TMP275 (U4) e il termistore NTC (R9), in marrone in figura: questo garantisce un buon contatto termico, peraltro facilitato dalla deposizione localizzata di una pasta termo-conduttrice.

Figura 58. Fissaggio della batteria e del PCB all'interno della scatola

Per garantire la tenuta stagna in prossimità del foro di uscita del sensore MS sono state usate guarnizioni e o-ring, assemblate come in Figura 59 (in arancio l'o-ring, in verde le guarnizioni e in rosso due rondelle in acciaio inox).

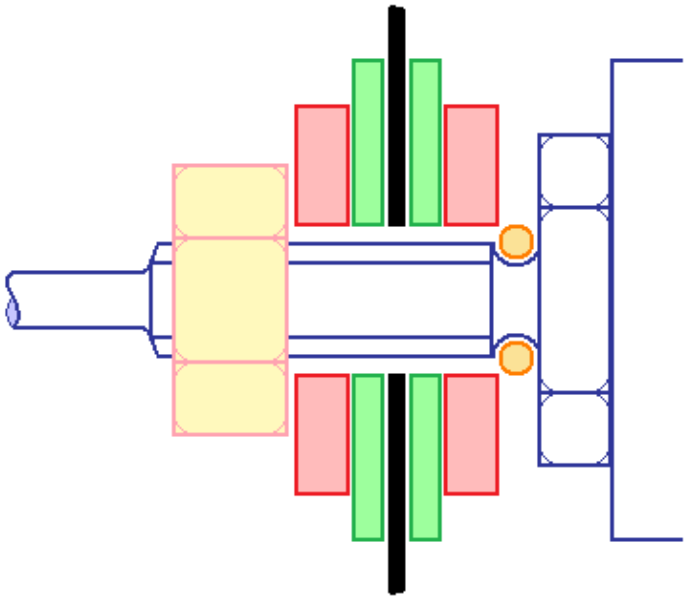


Figura 59. Realizzazione della tenuta stagna

Una volta assemblata la scheda e inserita nell'apposito alloggiamento della scatola è stato effettuato il collaudo della batteria, monitorando i suoi parametri elettrici e termici durante una fase di carica di 24 ore, memorizzando istante per istante i dati di tensione, corrente, carica e temperatura della batteria, con l'applicazione descritta che sarà descritta nel paragrafo 4.5.

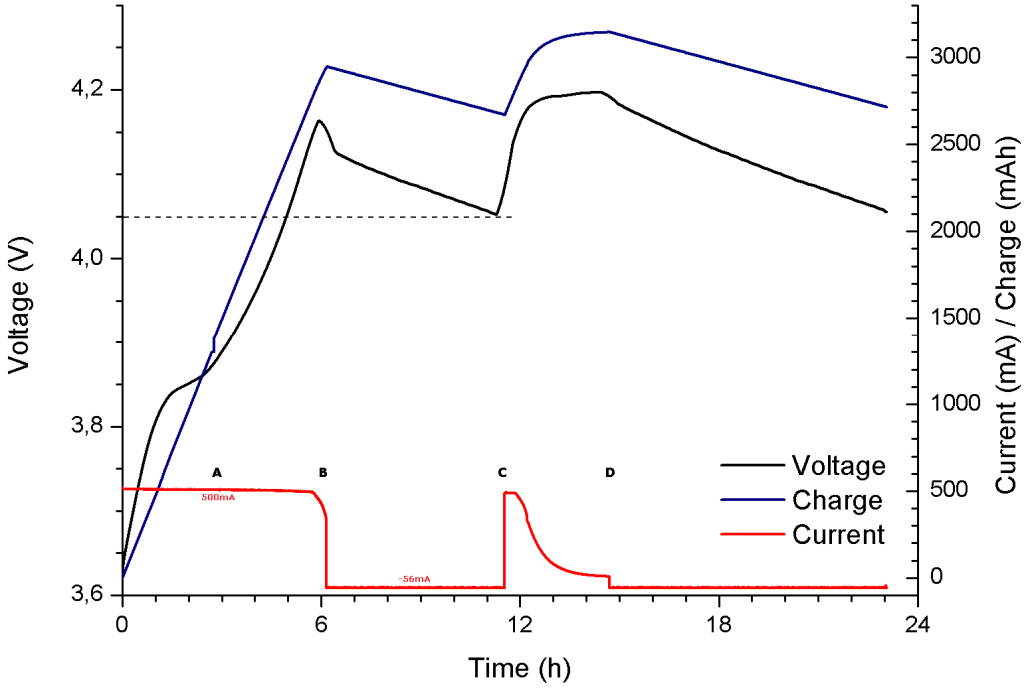


Figura 60. Carica della batteria

La ricarica inizia con la batteria precedentemente scaricata fino ad una tensione di 3,6 V. Il dispositivo viene quindi collegato ad una porta USB. Il caricabatterie inizia un primo ciclo di ricarica, a corrente costante (500 mA). Dopo 3 ore scade il timer interno al caricabatterie (punto A in Figura 60). La fase di ricarica viene interrotta ma istantaneamente viene riavviata in quanto la tensione della batteria è inferiore alla soglia di 4,05 V per cui la batteria è considerata carica. Viene così avviato un nuovo ciclo di 3 ore. Dopo circa 2,5 ore (punto B) la tensione raggiunge il valore limite di 4,2 V: inizia la fase di ricarica a tensione costante. Il timer si interrompe dopo 30 minuti interrompendo la carica. In questo caso la tensione è superiore a 4,05 V e il caricabatterie rimane spento. Viene notificato al microcontrollore che adesso la batteria è carica. D'ora in poi la batteria si scarica con la corrente assorbita dal circuito (circa 60 mA). Quando la tensione scende nuovamente sotto 4,05 V viene avviato un nuovo ciclo di ricarica (punto C). In questo caso la tensione è già vicina al valore massimo e

dopo pochi minuti si passa alla fase a tensione costante. E' evidente come la corrente decresca via via che la batteria si ricarica. Quando la corrente scende sotto il 10 % (50 mA) della corrente di ricarica impostata, il caricabatterie segnala al microcontrollore che la ricarica è quasi completata. Trascorse tre ore dall'inizio del ciclo la ricarica si interrompe nuovamente. Il caricabatteria viene disattivato e viene segnalato il completamento della ricarica.

In figura non è riportato l'andamento della temperatura della batteria: questa non si è mai discostata più di 0,5 °C dalla temperatura ambiente. Il risultato della misura coincide con quello atteso: caricando la batteria con una corrente di C/6, ben inferiore alla corrente di ricarica supportata (C/2), si ottiene una migliore efficienza della ricarica e uno stress minore della batteria stessa, riflettendosi positivamente sul numero di cicli di carica/scarica che potranno essere supportati durante la vita operativa del dispositivo.

4.5. Intefaccia LabVIEW

Tutti i comandi riconosciuti dal sensore sono stati testati innanzitutto utilizzando un terminale per inviare e ricervere stringhe sulle porte seriali del PC. La connessione Bluetooth è gestita, in questa fase, ricorrendo alla porta seriale virtuale messa a disposizione dal sistema operativo per ogni dispositivo col quale è stato effettuato correttamente il pairing. Per effettuare un collaudo esaustivo del firmware sono state previsti invii di risposte ridondanti, indicanti le transizioni delle macchine a stati che vengono seguite e le operazioni effettuate a bordo della scheda. Una volta verificatane le correttezza, queste risposte sono state eliminate. Si procede quindi allo sviluppo di un applicazione per PC che gestisca il collaudo del dispositivo con un'interfaccia user-friendly. Questa applicazione, sviluppata in *LabVIEW*, è in grado di acquisire tutte le informazioni sullo stato del dispositivo. Tali informazioni sono ridondanti ai fini del'utilizzo del sensore nella rete di misura di bordi liberi ma sono tuttavia importanti nelle fasi di collaudo e messa a punto del sistema, nonché ogni volta in cui si effettua una revisione del dispositivo. E' inoltre possibile inviare al sensore i comandi di taratura, sia per quanto riguarda la lettura della batteria, sia per il sensore MS. In Figura 61 è riportato uno screenshoot della finestra principale dell'applicazione, durante l'esecuzione.

Con questa applicazione è inoltre possibile monitorare la temperatura della batteria e dell'ambiente interno alla scatola, nonchè di effettuare un semplice data-logging su file.

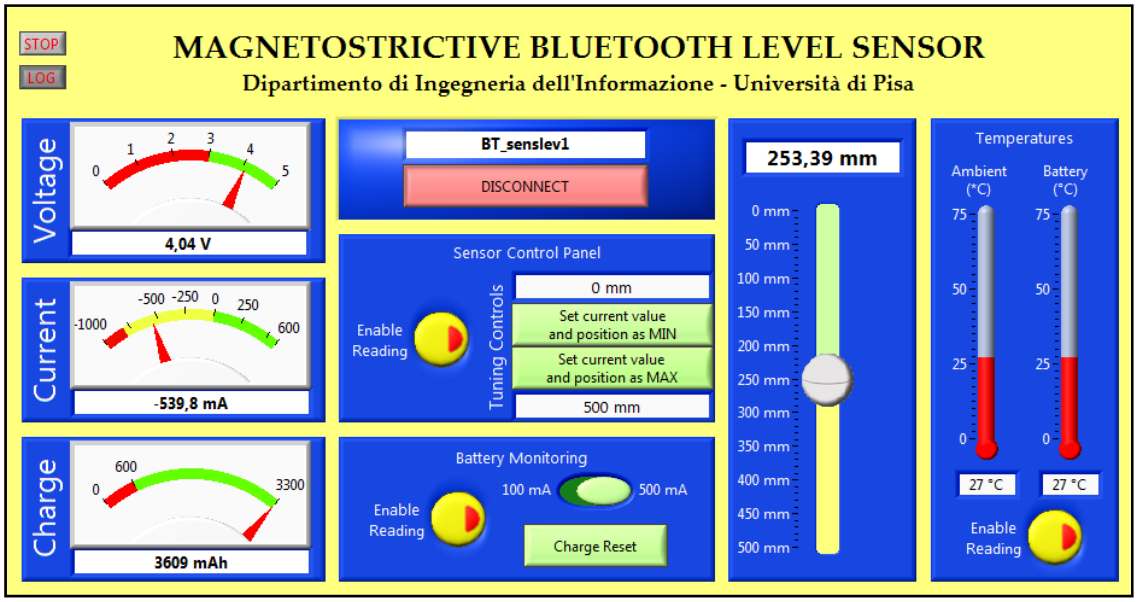


Figura 61. Pannello di controllo del sensore di livello (immagine da sostituire)

Per la controllare la connessione e lo scambio dei dati via Bluetooth si fa ricorso alle Vis fornite da *LabVIEW*, che si appoggiano direttamente alle system-call del sistema operativo. Il vantaggio principale di questo approccio è quello di poter indirizzare ogni misuratore di livello con il suo nome, e programmato una volta per tutte all'interno della radio Bluetooth Parani ESD-100, indipendente dalla macchina sulla quale sta girando l'applicazione. Tale approccio è lo stesso seguito nella realizzazione della seconda versione della libreria di Vis dell'inclinometro Bluetooth (descritta nel paragrafo 3.5).

Durante lo sviluppo di questa interfaccia di test è stato messo a punto un set di funzioni di lettura e di invio dei comandi da/verso il misuratore. Queste stesse VIs saranno utilizzate anche nell'applicazione di misura dei bordi liberi descritta nel capitolo 5.

5. Collaudo della rete di misura

Una volta verificato il funzionamento di un singolo misuratore di livello, ne sono stati realizzati altri due, costruttivamente e funzionalmente identico al primo. Le tre unità sono identificate esclusivamente attraverso il nome assegnato alla radio Bluetooth e/o l'indirizzo fisico MAC della stessa.

L'applicazione che controlla la rete di misura, denominata FBN (FreeBoard Network) Manager, è stata sviluppata in ambiente *LabVIEW*. L'interfaccia grafica è stata pensata per fornire al tecnico misuratore tutti i comandi essenziali per effettuare la misura automatizzata, nel modo più semplice e veloce possibile.

Si è cercato inoltre di rendere i controlli compatibili con un dispositivo portatile dotato di touch-screen, controllato sia mediante apposito stilo, sia direttamente con il tocco delle dita: per questo motivo sono stati utilizzati pulsanti di grosse dimensioni, visualizzati in diversi colori per renderne più immediata la localizzazione anche in condizioni di illuminazione non ottimale.

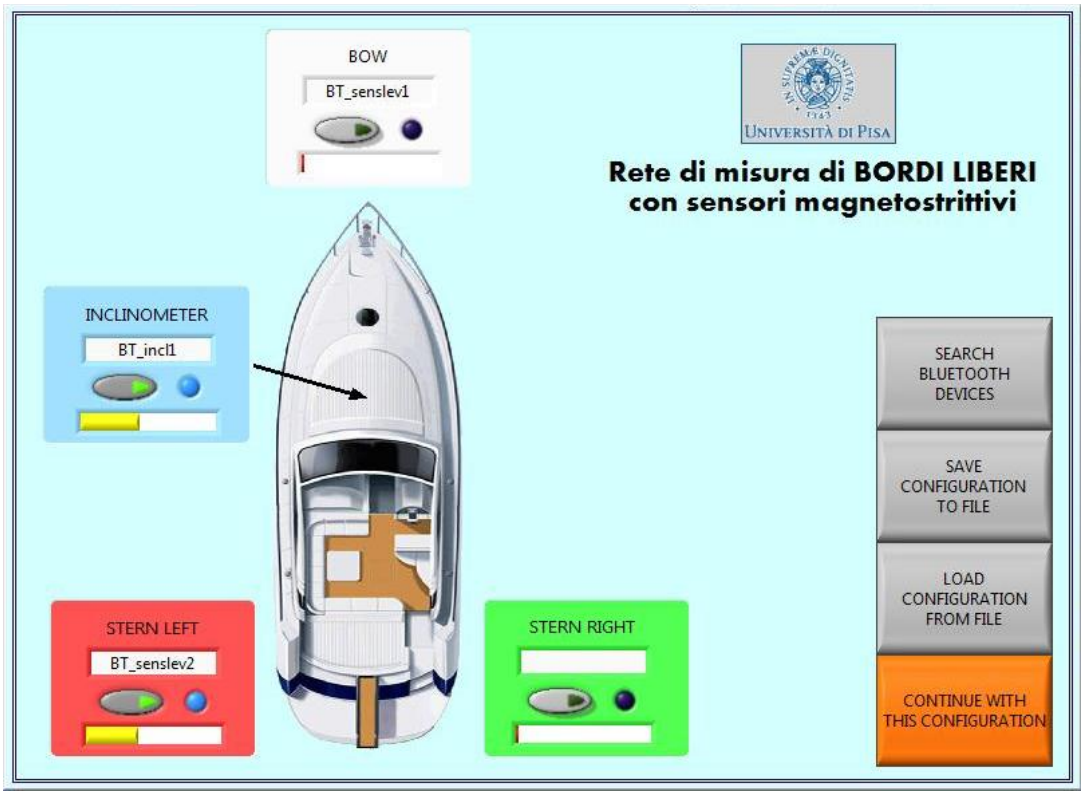


Figura 62. FBN Manager - schermata di connessione

All'avvio dell'applicazione viene presentata una schermata di configurazione della rete. In questa fase l'utente può effettuare la ricerca dei dispositivi Bluetooth disponibili nelle vicinanze. Quindi è possibile assegnare ad ogni posizione di misura il nome di un misuratore di livello. E' anche possibile selezionare un inclinometro Bluetooth da utilizzare. Queste operazioni possono essere velocizzate memorizzando la configurazione su file e quindi caricandola alla prova successiva.

Una volta selezionati i dispositivi, devono essere connessi agendo sugli appositi tasti. L'avvenuta connessione è notificata per mezzo di indicatori. Viene inoltre visualizzata l'informazione sullo stato della batteria di ogni dispositivo: è stato adottato un codice di colori che identifica i dispositivi in carica con una barra di colore verde altrimenti è usato il giallo per indicare un livello di carica sufficiente, il rosso per i dispositivi scarichi.

Completata la configurazione dei dispositivi è possibile procedere alla fase successiva, premendo il pulsante di conferma presente in basso a sinistra nella schermata. E' importante ricordare che alla pressione di questo pulsante viene abilitata l'alimentazione a 24 V su ogni misuratore di livello connesso, la quale assorbe una discreta potenza dalla batteria (circa 2 W). Nella schermata che si presenta vengono mostrati tre grafici in cui è riportata la posizione dei misuratori di livello collegati, in tempo reale. Anche in questo caso i colori utilizzati dovrebbero richiamare immediatamente la posizione del sensore: facendo riferimento ai colori dei fanali di navigazione, in particolare delle luci di via, si è assegnato il colore bianco al misuratore di prua, il rosso al misuratore sinistro, il verde al misuratore destro.

Se è stato connesso anche un inclinometro, i valori trasmessi sono visualizzati nella sezione dedicata, in alto a destra nella finestra. Qui è disponibile un pulsante che permette di effettuarne la calibrazione. Premendolo, vengono acquisiti i valori di inclinazione per 5 secondi, quindi viene effettuata una media di questi dati: il valore ottenuto viene utilizzato come valore di zero. E' necessario effettuare questa operazione ogni volta in cui l'inclinometro viene posizionato su di una superficie per la quale non è garantito dal

costruttore dell'imbarcazione l'allineamento orizzontale. Una volta completata questa fase è possibile procedere alla misura vera e propria.

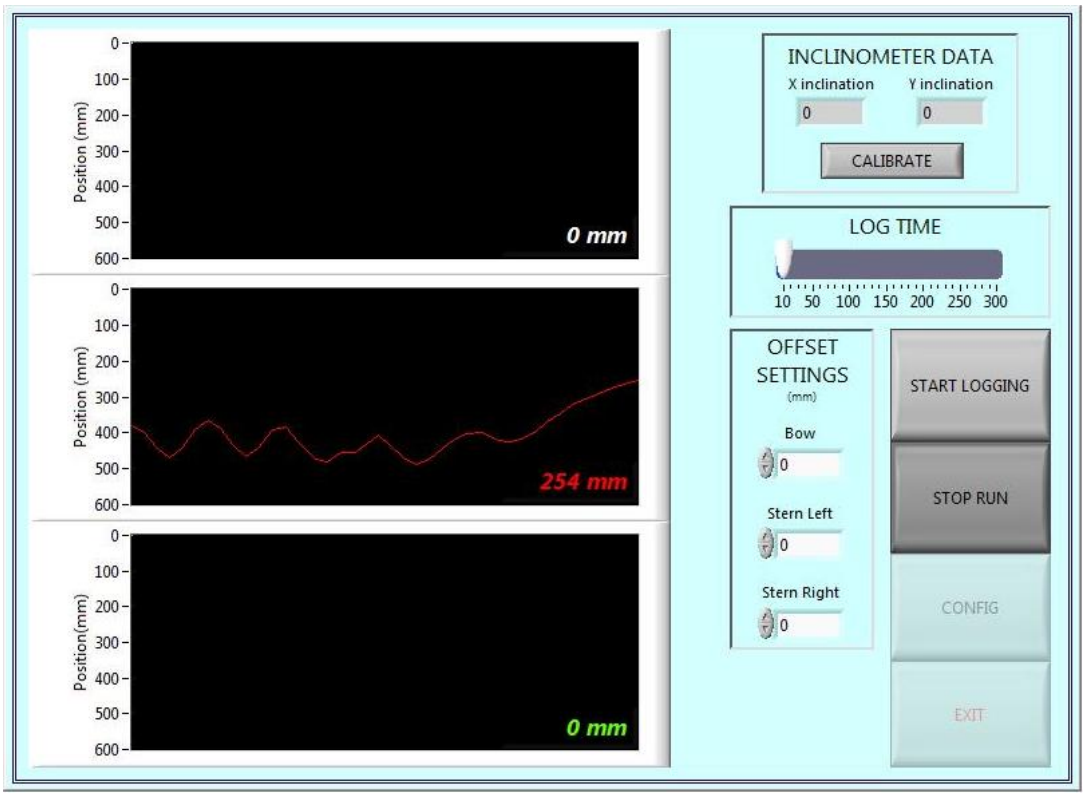


Figura 63. FBN Manager - schermata di visualizzazione

E' sufficiente premere l'apposito pulsante sulla destra della schermata per avviare la registrazione dei dati, dopo aver settato la durata della registrazione con il controllo a scorrimento. Verrà visualizzata una pagine dove i dati sono visualizzati durante l'acquisizione. Sia durante la misura che alla fine è possibile impostare una frequenza di taglio di un filtro passa-basso, usato per depurare i dati acquisiti dalle componenti di errore ad alta frequenza: il risultato del filtraggio è visualizzato direttamente sul grafico, attraverso una linea tratteggiata in sovraimpressione.

Alla fine dell'acquisizione vengono presentati i valori medi delle misure effettuate dai tre sensori. A questo punto si può scegliere se salvare i dati acquisiti su file, per successive elaborazioni, oppure tornare alla schermata precedente senza alcuna memorizzazione. E' possibile procedere ad una nuova misura.

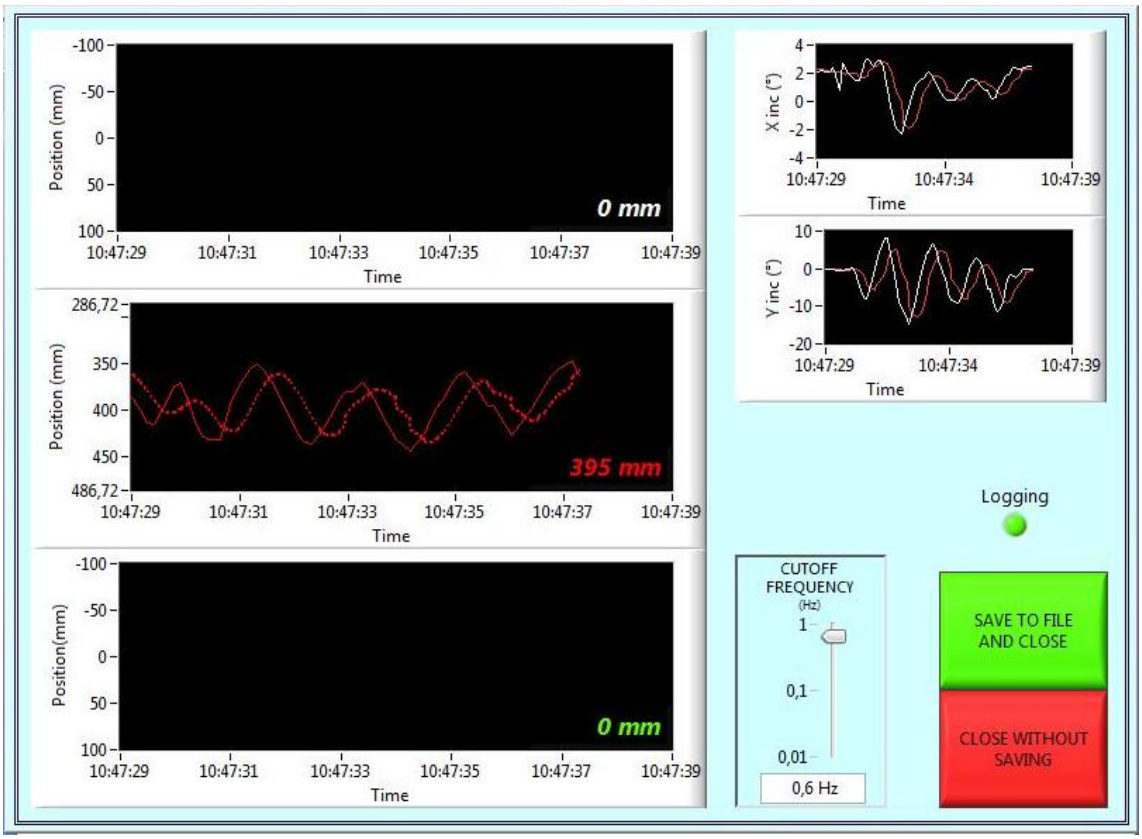


Figura 64. FBN Manager - schermata di salvataggio dei dati

Terminate tutte le operazioni si aziona il tasto di uscita: tutti i dispositivi wireless vengono scollegati e l'applicazione arrestata.

In Tabella 8 è riportato un esempio dei dati acquisiti con *FBN Manager*.

MISURA BORDI LIBERI DEL 23/06/2009 - ORE 10.17

BOW: 176,0			S.LEFT: 166,4		
BOW			STERN LEFT		
TIME	RAW	AVG	TIME	RAW	AVG
0	136,75	176,345	0,236	176,43	166,309
0	136,75	176,359	0,236	176,43	166,31
0,236	176,39	176,36	0,236	176,43	166,316
0,236	176,39	176,357	0,236	176,43	166,326
0,236	176,39	176,353	0,371	176,43	166,34
0,482	176,39	176,35	0,371	176,43	166,358
0,482	176,39	176,347	0,775	129,1	166,375
0,868	193,25	176,346	0,775	129,1	166,381
…	…	…	…	…	…

Tabella 8. Esempio di dati estratti con FBN Manager

L'applicazione è strutturata in due processi concorrenti non sincronizzati: un processo si occupa di effettuare le acquisizioni dei dati, l'altro si occupa della visualizzazione.

Ai due processi sono assegnate priorità diverse: viene privilegiato il processo di acquisizione, che è controllato tramite alcuni flag dal processo di interfaccia. Ad ogni dato acquisito viene associato il timestamp di macchina al momento dell'acquisizione. In questo modo il ciclo del processo di visualizzazione può essere eseguito a frequenza diversa senza perdere informazioni sulla temporizzazione dei dati, ai quali può accedere per mezzo di variabili condivise.

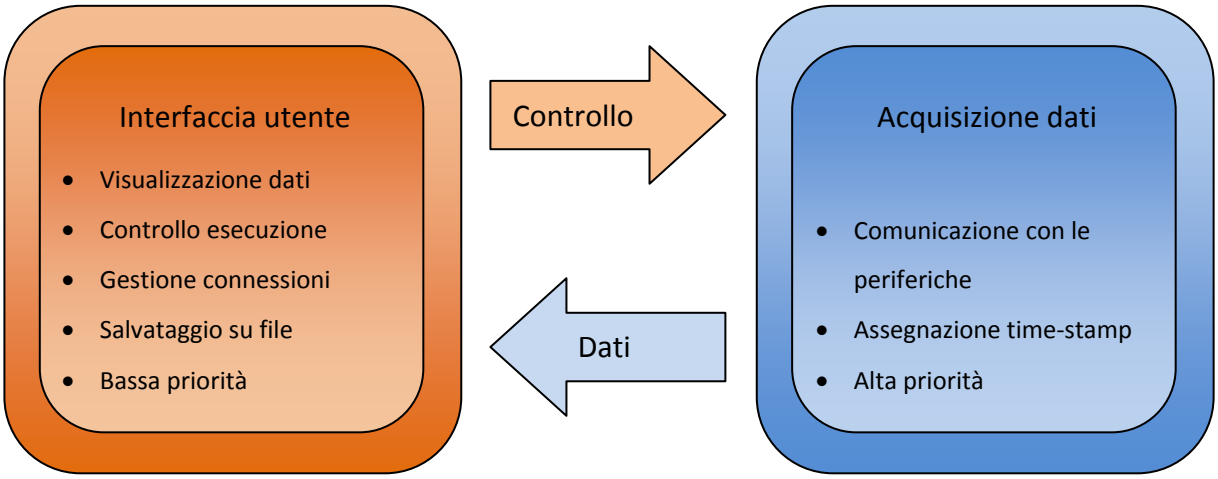


Figura 65. Struttura FBN Manager

E' importante notare che con questa strategia non si garantisce che per ognuno dei quattro nodi della rete si abbia un'acquisizione contemporanea, ma si tiene traccia di questa asincronia, riferendo ogni campione all'istante in cui è stato acquisito.

La modularità di questa architettura permette inoltre di intervenire sulla modalità di acquisizione dei dati indipendentemente dal processo di interfaccia. E'possibile sostituire completamente il processo di acquisizione secondo necessità, utilizzando ad esempio un modulo real-time, oppure un hardware dedicato in grado di effettuare acquisizioni ad alta velocità.

La rete di misura è stata inizialmente testata in laboratorio, verificando le risposte ai comandi e il corretto funzionamento dell'applicazione. In queste prove ci si è scontrati in un

problema legato alla gestione del protocollo SPP Bluetooth: questo introduce un overhead non trascurabile, in termini temporali, soprattutto per trasmissioni di pochi byte. Il massimo data-rate ottenibile.

Sono state effettuate quindi alcune prove sul campo, presso le banchine di Riva - Ferretti S.p.A. nel porto di La Spezia, a bordo di un Riva 86 in costruzione. Il posizionamento dei sensori è stato effettuato mediante aste telescopiche la cui lunghezza è stata misurata prima della sistemazione fuoribordo.

Figura 66. Sensori in posizione a bordo del Riva 86

Sono state effettuate diverse acquisizioni utilizzando l'applicazione *FBN Manager*. Durante le prove è stato riscontrato un problema nella connessione ai dispositivi: a causa della posizione delle radio bluetooth (molto basse sul'acqua) non è stato possibile individuarle tutte e quattro contemporaneamente. A seconda della posizione dell'unità centrale un dispositivo non riusciva a comunicare con sufficiente affidabilità, perdendo una quantità di dati significativa nella trasmissione. Tuttora sono in corso studi per evidenziare le cause del problema e trovarne una soluzione.

E' stata raccolta comunque una quantità di dati sufficiente spostando ogni volta il computer portatile e connettendo alternativamente unità diverse. Dai dati acquisiti possono già essere estratte informazioni significative sull'assetto dell'imbarcazione.

In Figura 67 sono riportati in un grafico i valori di bordo libero di poppa (destro e sinistro) in funzione del tempo. In basso è riportata anche la lettura dell'inclinometro. Risulta evidente la correlazione che esiste tra le due tracce: lo scafo segue il movimento delle onde agendo da filtro passa-basso, a causa della sua inerzia che elimina le variazioni più veloci e introduce un ritardo dell'ordine del secondo sulla risposta.

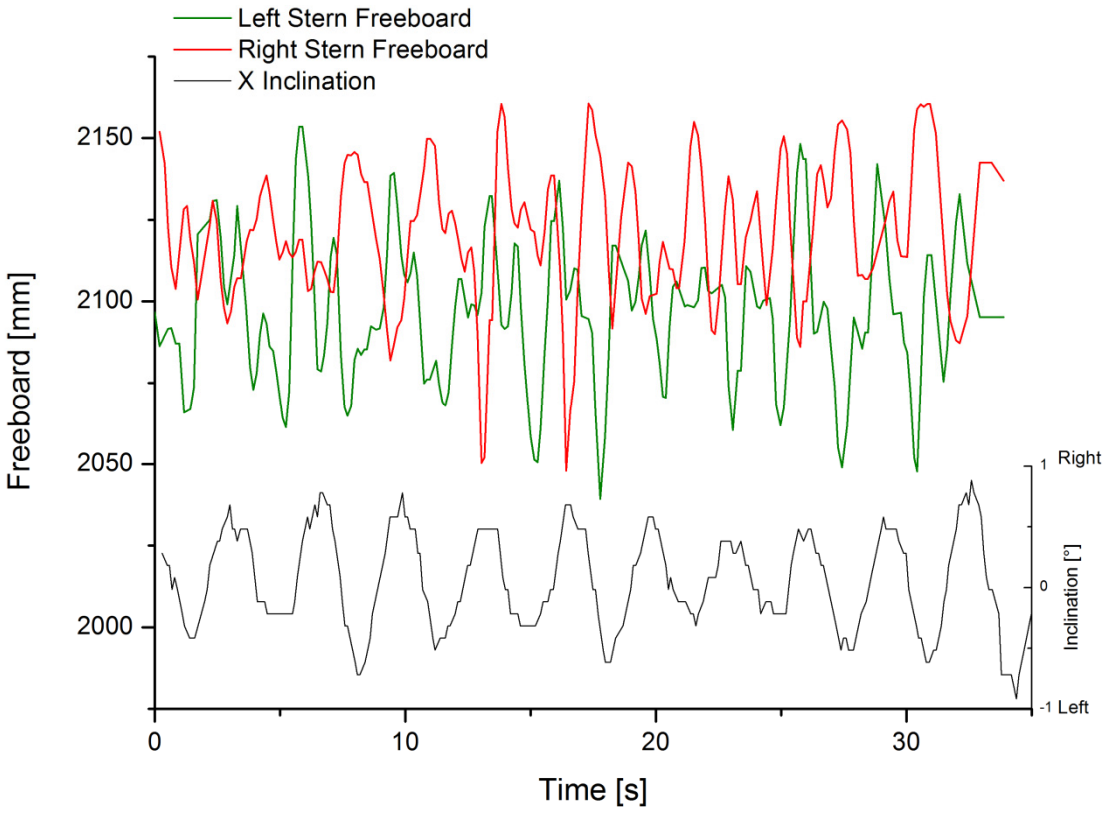


Figura 67. Acquisizione dei bordi liberi di poppa e dell'inclinazione

I bordi liberi misurati (2490 mm per la prua, 2096 mm per la poppa sinistra e 2124 per la poppa destra) sono praticamente coincidenti (scarti inferiori a 5 mm) con quelli misurati con il singolo sensore con uscita CAN-Open e corrispondono a quelli attesi dai progettisti dell'imbarcazione. La misura è fortemente influenzata dalla disposizione dei pesi a bordo, al punto che una sola persona che si sposti sul lato dritto o sinistro dell'imbarcazione provoca una sensibile variazione di livello, correttamente rilevata dai sensori (ad esempio, dal grafico

e dai risultati appena presentati si evince che l'imbarcazione è leggermente sbandata a sinistra).

Il reparto collaudi di Ferretti S.p.A. ha dimostrato grande interesse per i dati raccolti e per le potenzialità che il sistema è in grado di offrire, incoraggiandone ulteriori sviluppi per migliorarne l'affidabilità e la semplicità di utilizzo.

6. Conclusioni

In questo lavoro di tesi è stato sviluppato un sistema automatizzato per la misura di precisione dei bordi liberi di un'imbarcazione da diporto, pensato per semplificare il lavoro del tecnico stazzatore e sostituire la misura manuale mediante asta metrica. Il sistema sfrutta una rete senza fili, composta da tre misuratori di livello, un inclinometro, un'unità centrale.

I tre misuratori di livello sono stati completamente progettati, sviluppati e collaudati in questo lavoro di tesi. É stato necessario costruire una scheda (PCB) di interfaccia, in grado di collegarsi ad un sensore di livello industriale realizzato da un azienda esterna, di gestire correttamente lo scambio dei dati, di interfacciarsi verso un PC tramite collegamento senza fili Bluetooth. Il sistema è alimentato a batteria e dispone di caricabatteria integrato. Le soluzioni circuitali scelte permettono di raggiungere un'ottima autonomia della batteria (normalmente è possibile effettuare oltre 60 misure senza necessità di ricarica). L'elevata risoluzione del sensore scelto (46 µm) è evidentemente di molto superiore a quella ottenibile con la misura manuale che la rete di misura si propone di sostituire.

La singola unità di misura è stata inizialmente collaudata singolarmente, analizzando il funzionamento del dispositivo in tutte le sue fasi. Per facilitare le operazioni di diagnosi e intervento sui guasti, il firmware è programmato in modo da notificare su un PC, collegato tramite Bluetooth, eventuali errori o anomalie nel funzionamento.

Utilizzando batterie ai polimeri di litio ad alta densità di energia (necessaria per alimentare il sensore utilizzato) è stato necessario affrontare problematiche legate alla gestione dell'energia e al controllo termico di questi elementi. Un utilizzo improprio di questi accumulatori può provocare danni al sistema che alimentano ma, nei casi più gravi, anche all'utente che lo utilizza. E' stato effettuato un accurato collaudo delle batterie una volta connesse al sistema, monitorandone i parametri elettrici e termici durante la carica e la scarica.

Ai misuratori di livello è stato affiancato un inclinometro Bluetooth, anch'esso alimentato a batteria, inizialmente sviluppato per essere integrato nel progetto Black-Box per Ferretti

S.p.A. Per questo dispositivo sono state curate le fasi di sviluppo del firmware e di revisione del circuito per la successiva produzione, nonché di collaudo e di sviluppo di software. Uno degli esemplari prodotti è attualmente utilizzato nelle prove di collaudo di Luxury Yacht Ferretti. Nelle successive release può essere interessante ricercare una soluzione che offra una migliore risoluzione di misura, utile soprattutto nella validazione della misura di bordi liberi.

L'insieme dei dispositivi wireless viene gestito da un PC su cui gira un applicativo *LabVIEW*, sviluppato ad hoc per questa funzione. Un'interfaccia grafica guida il tecnico stazzatore alla configurazione della rete, quindi visualizza in tempo reale i tutti i dati misurati e permette di memorizzarli su file per successive elaborazioni.

Altro software è stato prodotto per gestire a basso livello ogni singolo dispositivo permettendo di verificare l'accuratezza delle misure, di diagnosticare guasti e malfunzionamenti, di eseguire tarature sulle misurazioni e sui parametri delle batterie.

Fondamentale nello sviluppo del lavoro è stata l'esperienza maturata nel campo dell'acquisizione ed il processing dei dati da sensori, durante lo sviluppo del Progetto Black Box. Per questo sistema sono state create e/o sviluppate alcune applicazioni, quale ad esempio un sistema di misura del raggio di virata dell'imbarcazione sfruttando i dati forniti da un ricevitore GPS.

Il sistema è stato collaudato a bordo di un'imbarcazione (Riva 86) in costruzione presso i cantieri Riva – Ferretti S.p.A. di La Spezia. Sono state riscontrate difficoltà nella connessione contemporanea di tutti i dispositivi della rete, per cause che sono tuttora investigate. Nonostante questo, sono state effettuate diverse misure connettendo alternativamente diversi gruppi di sensori: i risultati ottenuti coincidono con quelli prodotti da altre misure precedentemente effettuate dal cantiere stesso. Inoltre è stato possibile estratte altre informazioni post-processando i dati acquisiti. Questi danno informazioni sull'assetto dinamico dell'imbarcazione e sulla risposta di questa alle increspature dell'acqua e alle piccole onde. Il cantiere ha dimostrato da subito interesse nei risultati ottenuti, che vanno oltre le loro aspettative.

Il successo ottenuto nella prova incoraggia l'investimento di risorse per portare avanti il progetto e mettere a punto aspetti che non sono stati trattati in questo lavoro.

In particolare è necessario effettuare uno studio per definire e realizzare un sistema di ancoraggio meccanico per i misuratori di livello, che sia allo stesso tempo robusto e affidabile ma che offra facilità e velocità di installazione a bordo, senza per questo inficiare la precisione della misura effettuata.

Deve essere definita univocamente una procedura di taratura dei misuratori di livello, affidabile e facilmente ripetibile. Un'alternativa all'utilizzo di un'asta metrica può essere quella di effettuare una misura di volume di un liquido introdotto all'interno di un cilindro di geometria nota nel quale è inserita l'unità di misura.

Inoltre devono essere approfonditi gli aspetti legati al post-processing dei dati ottenuti dalla misura, per fornire all'operatore dati utili per i fini per cui la misura è stata effettuata: a seconda che si tratti, ad esempio, di una misura destinata alla validazione del progetto di uno scafo o alla messa a punto di un'imbarcazione a vela da competizione devono essere estratte indicazioni diverse.

Gli aspetti legati alla comunicazione Bluetooth possono essere indagati per superare la limitazione imposta sul massimo data-rate dal protocollo SPP. Anche la realizzazione di una rete cablata (su bus RS-485) può essere altrettanto interessante: può fornire vantaggi significativi rispetto alla misura manuale, riducendo però i costi realizzativi rispetto alla soluzione wireless.

In questo lavoro, come nelle precedenti collaborazioni con Ferretti S.p.A. (progetto BlackBox) è stata maturata una discreta esperienza nell'acquisizione di dati da sensori e periferiche diverse. Sono stati evidenziati pregi e limitazioni di architetture basate su S.O. general-purpose e non real-time. L'evoluzione di questi progetti potrebbe portare alla realizzazione di una piattaforma di acquisizione dati ad alta velocità, multiprotocollo e real-time, sulla quale costruire, successivamente, software ad-hoc per le varie applicazioni.

Bibliografia

[1] Emanuele Marraccini, *Sistema elettronico per l'acquisizione e la memorizzazione dei dati di bordo nel collaudo di uno Yacht da Crociera,* Tesi di Laurea.

[2] NMEA data - GPS Information, http://www.gpsinformation.org/dale/nmea.htm.

[3] Giuseppe Iannaccone, *Appunti del corso di Infrastrutture per l'Habitat – Bluetooth, a.a. 2003-2004.*

[4] Analog Devices, *ADIS16201 Datasheet.*

[5] Sena, *Parani ESD-100 Datasheet.*

[6] Linear Technology, *LTC4053 Datasheet.*

[7] One Wire (Wikipedia), http://en.wikipedia.org/wiki/One_wire.

[8] Future Technology Devices International Ltd, *FT232 Datasheet.*

[9] Serial Pheripheral Interface Bus (Wikipedia), http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus.

[10] Texas Instruments, *TPS2010-2013 Datasheet.*

[11] Atmel Corporation, *ATtiny2313 Datasheet.*

[12] SensorLand - How It Works, http://www.sensorland.com/HowPage024.html.

[13] Vincenzo Villa, Vincenzo Villa Home Page, http://www.vincenzov.net.

[14] Linear Technology, *LT1371 Datasheet.*

[15] Linear Technology, *LTC1480 Datasheet.*

[16] DS Europe, *Istruzioni per l'installazione e l'uso dei sensori magnetostrittivi - Serie PC - Ver. 1, Rev. 5.*

[17] Pascal Stang, *AVR-lib [AVR µC Library]*

Ringraziamenti

Desidero ringraziare calorosamente tutte le persone che in questi mesi hanno contribuito alla realizzazione di questo lavoro.

Per primo l'Ing. Marraccini, che ha iniziato a lavorare sul progetto Ferretti prima di me: la sua esperienza e conoscenza del sistema è stata indispensabile.

Quindi ringrazio tutto il laboratorio Testing del Dipartimento di Ingegneria dell'Informazione. In particolare l'Ing. Baronti che ha pazientemente seguito i miei errori e anche qualche disastro, e ha sempre saputo stimolare la curiosità ed interesse nello svolgere il mio lavoro.

Grazie al Prof. Roncella, sempre presente in laboratorio, disponibile a consigliarmi sulle scelte da fare (anche se non strettamente legate all'ingegneria elettronica), a sperimentare nuove soluzioni, e anche ad effettuare un esame autoptico su una batteria ai polimeri di litio prima di farla bollire in un secchio d'acqua.

Infine, Grazie al Prof. Saletti, che mi ha seguito da quando ho iniziato questo lavoro, nell'Ottobre del 2008, e allo stesso tempo mi ha lasciato libero di fare determinate scelte. Ho avuto così la possibilità di confrontarmi con i miei colleghi, di mettere in opera le soluzioni che ritenevo migliori, di rendermi conto dei miei errori. Dunque lo ringrazio per l'opportunità di crescita intellettuale che mi ha offerto.

Grazie anche a coloro che non ho citato ma che hanno dato, in un modo o nell'altro, un loro contributo, senza il quale questi risultati non sarebbero stati ottenuti.

Grazie,

Gabriele Fantechi

Appendice A. Indice delle figure

Appendice B. Firmware inclinometro

SensorDef.h

```
//ADIS Register Addresses
#define SUPPLY_OUT   0x02
#define XACCL_OUT    0x04
#define YACCL_OUT    0x06
#define AUX_ADC      0x08
#define TEMP_OUT     0x0A
#define XINCL_OUT    0x0D
#define YINCL_OUT    0x0F
#define XACCL_OFF    0x10
#define YACCL_OFF    0x12
#define XACCL_SCALE  0x14
#define YACCL_SCALE  0x16
#define XINCL_OFF    0x18
#define YINCL_OFF    0x1A
#define XINCL_SCALE  0x1C
#define YINCL_SCALE  0x1E
#define ALM_MAG1     0x20
#define ALM_MAG2     0x22
#define ALM_SMPL1    0x24
#define ALM_SMPL2    0x26
#define ALM_CTRL     0x28
#define AUX_DAC      0x30
#define GPIO_CTRL    0x32
#define MSC_CTRL     0x34
#define SMPL_PRD     0x36
#define AVG_CNT      0x38
#define PWR_MDE      0x3A
#define STATUS       0x3C
#define COMMAND      0x3E
```

main.h

```
volatile unsigned char UART_data;
enum state {POWERDOWN,USB,WAITING,BLUETOOTH,BOOT} state_var;

/*
sono definiti i seguenti STATI:

POWERDOWN--> nesuna comunicazione, uC in power-down
BOOT--> stato inizio programma - mostra stato batteria
WAITING--> in attesa di connessione Bluetooth
BLUETOOTH-->connessione Bluetooth instaurata
USB--> attivata modalità USB
*/

//---COUNTER0 definizione TOP VALUE---
#define F_TICK 100UL // Tick freq=100Hz (Tick period=10ms)
#define PRESC 1024UL// PRESCALER Fckio/1024
#define TIME_TOP_VALUE ((F_CPU/(PRESC * F_TICK))-1) //OCR0A Compare Register Value

//---INPUT NAMES DEFINES---
#define PUSH_IN PIND2
#define USB_CONNECT PIND3
#define BLUETH_CONNECT PINB1
#define NOT_PD PB0

//---INPUT DATA READING DEFINES---
#define SUPPLY_STAT  !(PIND & (1 << USB_CONNECT)) //USB connect data reading define
#define PUSHBUTTON  !(PIND & (1 << PUSH_IN)) // Pushbutton data reading define
#define BT_CONN  !(PINB & (1 << BLUETH_CONNECT)) // Bluetooth connection data reading define
#define SENSOR_RESET PB2
#define MUX_TO_BT          PORTB &= ~(1<<PB3); //USART Routing to BT
#define MUX_TO_USB         PORTB |= (1<<PB3);  //USART Routing to USB

//---VARIABLES---
volatile unsigned char push_done;//0=tasto non premuto, 1= tasto premuto
volatile uint16_t timeout; //temporizzazione di timeout
volatile unsigned char tick_system; //flag di sincronizzazione
volatile long int press_count;     //contatore per rilevamento della pressione prolungata
volatile unsigned char discharge;  //0=timeout attivato, 1=modalità di scarica rapida
```

main.c

```
#include <avr/io.h>
#include <avr/interrupt.h>
```

```
#include <avr/sleep.h>
#include "spi_via_usi_driver.h"
#include "SensorDef.h"
#include "delay.h"
#include "led.h"
#include "one_wire_driver.h"
#include "main.h"

const uint16_t version=1;


//---Interrupt service routines

//Pushbutton INT0 interrupt service routine
ISR(INT0_vect){
	press_count=0; //clear long press counter
	LED_ON;
	_delay_us (1400);//debounce
	while ((!(PIND & (1 << PUSH_IN)))) {
		press_count++; //long press counter ++
		if (press_count>400000) { //long press detected
			state_var=POWERDOWN;
			LED_OFF;
			}
		}
		_delay_us (1400);//debounce
		push_done=1;//setting flag
	}

//TIMER0 interrupt service routine
ISR(TIMER0_COMPA_vect){
	timeout++; //increase timeout timer
	tick_system=1; //set the sync. flag
	}

//---MAIN---
int main (void){
	//Set the ADIS /reset ad OUTPUT (PB2) and set it high
	DDRB  |=      (1<<DDB2); //write 1 in DDB2
	PORTB |=    (1<<PB2); // set high locig level

	// Set the e /PD pin as OUTPUT(PB3)
	DDRB  |=      (1<<DDB3)||(1<<DDB0);

	// Set the BT_CONN pin as INPUT(PB1)
	DDRB  &=      ~(1<<DDB1);//write 0 in DDB1
	PORTB |=     (1<<PB1);// activate pull-up resistor

	// Set the Push_IN pin as INPUT(PD2)
	DDRD  &=      ~(1<<DDD2);// write 0 in DDD2
	PORTD |=     (1<<PD2);// activate pull-up resistor

	// Set the Supply_STAT pin as INPUT(PD3)
	DDRD  &=      ~(1<<DDD3);//write 0 in DDD3
	PORTD &=      ~(1<<PD3);// deactivate pull-up resistor

	// Set the Charge select pins as OUTPUTs(PD4, PD5)
	DDRD  &=      ~((1<<DDD4)|(1<<DDD5));//Battery managment operations - HighZ
	PORTD &=     ~((1<<PD4)|(1<<PD5));// deactivate pull-up resistor (100mA charge)

	// Set the LED Driving port as OUTPUT(PD6)
	DDRD  |=      (1<<DDD6);//write 1 in DDD6

	spiX_initmaster(); //SPI communication initializing

	USART_Init();  //USART communication initializing

	state_var = POWERDOWN; //set start state as "powerdown"

	set_sleep_mode (SLEEP_MODE_PWR_DOWN);// set sleep mode as power-down

	//-------------Interrupts management-------------

	//Enable External Interrupt INT0
	MCUCR  &=  ~(1<<ISC00); //Interrupt 0 sense control:
	MCUCR  &=  ~(1<<ISC01); //  the low level of INT0 generates an interrupt request
	GIMSK  |=     (1<<INT0);  //INT0 interrupts enabled

	//setting TMR0 for CTC Mode
	TCCR0A |= (1<<WGM01);
	TCCR0A &= ~(1<<WGM00);
	TCCR0B |= (1<<CS02)|(1<<CS00);
```

```
TCCR0B &= ~(1<<WGM02);
TCCR0B &= ~(1<<CS01);
OCR0A = TIME_TOP_VALUE;
TIMSK |= (1<<OCIE0A);  //  Timer interrupts enabled

sei(); // Enable global interrupts

//-----------------------------------------------

DS2751_write (0x31,0b00000010); //Setting Offset Blanking in battery gauge


//-------------Infinite loop---------------------
while(1){
       if (tick_system){
              tick_system=0;
              switch (state_var) {

                     //POWERDOWN state
                     case POWERDOWN:{
                            LED_OFF;
                            DDRD  &= ~((1<<DDD4)|(1<<DDD5));//Battery management operations - HighZ
                            while (!(PIND & (1 << PUSH_IN)));//waiting for button's release
                            delay_us (1400);//debounce
                            PORTB |= (1<<NOT_PD); //BT and ADIS off
                            PORTB &= ~(1<<SENSOR_RESET); //ADIS /reset low
                            sleep_enable();// Set the SE (sleep enable) bit.
                            sleep_cpu();// Put the device in power-down mode
                            //---sleeping...---
                            //--- wake up ---
                            sleep_disable();// Clear the SE (sleep enable) bit.
                            timeout=0;// reset counter before going into WAITING state
                            PORTB |= (1<<SENSOR_RESET); //ADIS /reset high
                            PORTB &= ~(1<<NOT_PD); //BT and ADIS on
                            state_var = BOOT; //waiting connection state
                            //select LED blinking sequence
                            if(DS2751_read (C_REG)<984) LED_toggle_init(LED_ON_TIME_BAT_LOW_CHG,
                                                                        LED_PULSE_LEN_BAT_LOW_CHG)
                                                                        ;//Full charge
                            else LED_toggle_init(LED_ON_TIME_BAT_FULL_CHG,
                                                                        LED_PULSE_LEN_BAT_FULL_CHG
                                                                        );//Low charge
                            spiX_write(AVG_CNT, 0x08); //Sensor AVG settings
                            discharge=0; //clear fast discharge flag
                            break;
                            }

                     //BOOT state
                     case BOOT:{
                            if (timeout > 1200) {//Boot time expired
                                   state_var = WAITING;
                                   timeout=0;//reset timeout counter
                                   }
                            LED_toggle_task(); //LED blinking routine
                            push_done=0; //clear button flag
                            break;
                            }

                     //WAITING BT connection state
                     case WAITING:{
                            LED_ON;
                            MUX_TO_BT; // UART connected to BT radio
                            if (push_done==1){//if pushbutton is pressed -->USB
                                   push_done=0;
                                   LED_OFF;
                                   LED_toggle_init(LED_ON_TIME_USB,LED_PULSE_LEN_USB);
                                   state_var = USB;
                                   }
                            if (BT_CONN){//if BT connected --> BLUETOOTH
                                   LED_OFF;
                                   LED_toggle_init(LED_ON_TIME_BLUETOOTH,LED_PULSE_LEN_BLUETOOTH);
                                   state_var = BLUETOOTH; //get bluetooth connection -->BT connection
                                                                        state
                                   }

                            //if fast discharge flag is not set and timeout timer expires...
                            if ((timeout>60000)&&(!SUPPLY_STAT)&&(!discharge)){
                                   state_var = POWERDOWN;
                                   break;
                                   }
                            }
```

```
                        //BLUETOOTH connection state
                        case BLUETOOTH:{
                               push_done=0;
                               LED_toggle_task();
                               if (!(BT_CONN)) { //if BT disconnected --> WAITING
                                      timeout=0;// reset counter
                                      push_done=0;
                                      state_var = WAITING;
                                      }
                               break;
                               }

                        //USB connection state
                        case USB: {
                               MUX_TO_USB; // UART connected to FTDI
                               LED_toggle_task();
                               if ((push_done==1)||((!SUPPLY_STAT))){ //if button is pressed or USB cable
                                                                              is disconnected
                                      timeout=0;// reset counter before going into WAITING state
                                      push_done=0;
                                      state_var = WAITING; //if pushbutton is pressed -->waiting connection
                                                                              state
                                      }
                               break;
                               }

                        default : break;

                        }//end switch

                //-------------Executing commands----------------
                if(UCSRA & (1<<RXC)) {
                        UART_data = UDR;
                        switch (UART_data) {

                                //Charging with 100mA
                                case 0x36: {DDRD  &=~((1<<DDD4)|(1<<DDD5)); break;}

                                //Charging with 500mA
                                case 0x37: {DDRD  |=     ((1<<DDD4)|(1<<DDD5));    break;}

                                //( D - set discharge flag - will be cleared going into POWERDOWN)
                                case 0x44: {discharge=1;   break;}

                                //(E - Reset charge accumulator)
                                case 0x45: {DS2751_write(C_REG,0x00);DS2751_write(C_REG+1,0x00); break;}

                                //Read V
                                case 0x31: {USART_Transmit(DS2751_read (V_REG)); break;}

                                //Read I
                                case 0x32: {USART_Transmit(DS2751_read (I_REG)); break;}

                                //Read C
                                case 0x33: {USART_Transmit(DS2751_read (C_REG)); break;}

                                case 0x34: {
                                       USART_Transmit(spiX_read(XINCL_OUT) & 0x0FFF);
                                       USART_Transmit(spiX_read(YINCL_OUT) & 0x0FFF);
                                       break;
                                       }

                                case 0x35: {
                                       USART_Transmit(spiX_read(XACCL_OUT) & 0x3FFF);
                                       USART_Transmit(spiX_read(YACCL_OUT) & 0x3FFF);
                                       break;
                                       }

                                case 0x38: {USART_Transmit(spiX_read(STATUS) & 0x01FF); break;}

                                case 0x56: {USART_Transmit(version); break;}

                                default: break;

                                } //end switch

                        }//end if (UART)

                }//end if (tick_system)

        } //end infinite while
```

```
      } //end main
```

led.h

```
//-------------------------LED status defines--------------------------------
#define LED_ON  PORTD &= ~(1<<PD6);// turn ON the led clearin PD6
#define LED_OFF  PORTD |= (1<<PD6);// turn OFF the led setting PD6

//------------------------ ms TO TICKs CONVERSION -----------------------------
#define LED_ON_TIME_IN_TICKS      (led_on_time * F_TICK)/1000
#define LED_PULSE_LEN_IN_TICKS    (led_pulse_len * F_TICK)/1000


//                         FOLLOWING TIMES ARE EXPRESSED IN ms

//-------------------------USB Connected--------------------------------------
#define LED_ON_TIME_USB              600
#define LED_PULSE_LEN_USB           700
//-------------------------Bluetooth Connected--------------------------------
#define LED_ON_TIME_BLUETOOTH         50
#define LED_PULSE_LEN_BLUETOOTH       2000
//-------------------------Battery Full Charge--------------------------------
#define LED_ON_TIME_BAT_FULL_CHG      500
#define LED_PULSE_LEN_BAT_FULL_CHG    1000
//-------------------------Battery Low Charge---------------------------------
#define LED_ON_TIME_BAT_LOW_CHG       50
#define LED_PULSE_LEN_BAT_LOW_CHG     100

//variable definitions
uint16_t led_on_time, led_pulse_len, LED_ticks_count;

//functions declarations
void LED_toggle_init(uint16_t led_on_time_in, uint16_t led_pulse_len_in);
void LED_toggle_task();
```

led.c

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include "led.h"
#include "main.h"

void LED_toggle_init(uint16_t led_on_time_in, uint16_t led_pulse_len_in){
       led_on_time = led_on_time_in;  //initialize on_time
       led_pulse_len = led_pulse_len_in; //initialize period
       LED_ticks_count = 0; //clear variables
       }

void LED_toggle_task() {
       if(LED_ticks_count == LED_PULSE_LEN_IN_TICKS) LED_ticks_count = 0;
       else LED_ticks_count++;
       if((LED_ticks_count < LED_ON_TIME_IN_TICKS)) LED_ON else LED_OFF;
       }
```

one_wire_driver.h

```
// Definizione registri battery gauge
#define V_REG 0x0C
#define I_REG 0x0E
#define C_REG 0x10

// Definizione routing
#define GaugePORT PORTB
#define GaugeDDR  DDRB
#define Gauge   4


// Dichiarazione funzioni
void one_wire_reset(void);
char one_wire_read_bit(void);
unsigned char one_wire_read_byte(void);
void one_wire_write_bit(unsigned short int bit);
void one_wire_write_byte(unsigned short int data);
void DS2751_write (char address,char data);
int DS2751_read (char address);
```

one_wire_driver.c

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include "delay.h"
```

```
#include "one_wire_driver.h"
#define W_COMMAND 0x6C
#define R_COMMAND 0x69
#define SN_ADDRESS 0xCC

/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Per l'utilizzo della funzione "_delay_us(ms)" si fa la include della libreria "delay.h", che
a sua volta richiede la presenza della libreria "delay_basic.h":sono state inserite entrambe
nella cartela del progetto.
Per un corretto uso della funzione di ritardo è necessario ottimizzare la compilazione almeno al
livello "-O1": questo garantisce che la funzione ritardi effettivamente di qunto richiesto, ma
tutto dipenderà  dalla particolare configuarazione hardware ivi implementata: non è garantita
la portabilità, va tenuto presente.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/


/*****************************************************************************************
Funzione di invio impulso basso di RESET della comunicazione 1-Wire e attesa segnale
di presenza del dispositivo slave.
Se il dispositivo è presente si riceve 0 sul bus, altrimenti si riceve 1.
La funzione ritorna il bit di presenza.
*****************************************************************************************/
void one_wire_reset(void){
       GaugeDDR  |=  (1<<Gauge);// setto in scrittura la linea 1-wire (Gauge=1)
       GaugePORT &=    ~(1<<Gauge);// pone a 0 la linea 1-wire
       _delay_us (480); // RITARDO di 480 us
       GaugePORT |=    (1<<Gauge);// pone a 1 la linea 1-wire
       GaugeDDR  &=  ~(1<<Gauge);// setto in lettura la linea 1-wire (Gauge=0)
       _delay_us (480);//ATTESA bit di presenza di 70us
       }

/*****************************************************************************************
Funzione di lettura di un bit da bus 1-wire.
Ritorna il bit letto. Si utilizza un time-slot di 65 us.
*****************************************************************************************/
char one_wire_read_bit(void){
       char result; //
       GaugeDDR  |=  (1<<Gauge);// scrittura sulla linea 1-wire (Gauge=1)
       GaugePORT &=    ~(1<<Gauge);// scrittura 0
       _delay_us (2);
       GaugeDDR  &=  ~(1<<Gauge);//rilascio della linea 1-wire (Gauge=0)
       GaugePORT |=    (1<<Gauge);  //attivazione pull-up
       _delay_us(8);
       result = (PINB & (1<<Gauge));// lettura linea
       _delay_us (55);
       return result;
       }

/*****************************************************************************************
Funzione di lettura di un byte da bus 1-wire.
Ritorna il byte letto. Si utilizza un time-slot di 65 us.
*****************************************************************************************/
unsigned char one_wire_read_byte(void){
       char loop;
       unsigned char result = 0;
       for (loop=0;loop<8;loop++){
              result >>= 1;//shifta a destra di 1 bit il risultato per renderlo pronto per il bit successivo
              if (one_wire_read_bit()) result |= 0x80;//se il bit letto è 1 setta l'LSB del risultato
              }
       return result;
       }

/*****************************************************************************************
Funzione di scrittura di un bit su bus 1-wire.
Scrive su bus 1-wire il bit passato come argomento.
*****************************************************************************************/
void one_wire_write_bit(unsigned short int bit) {
       if (bit){//se il bit da scrivere è 1
              GaugeDDR  |=  (1<<Gauge);// setta in scrittura la linea 1-wire (Gauge=1)
              GaugePORT &=    ~(1<<Gauge);//  pone a 0 la linea 1-wire
              _delay_us (15);
              GaugePORT |=    (1<<Gauge);// pone a 1 la linea 1-wire rilasciando il bus
              _delay_us (64);
              }
       else{//se il bit da scrivere è 0
              GaugeDDR  |=  (1<<Gauge);// mette in scrittura la linea 1-wire (Gauge=1)
              GaugePORT &=    ~(1<<Gauge);//  pone a 0 la linea 1-wire
              _delay_us (60);
              GaugePORT|= (1<<Gauge);// pone a 1 la linea 1-wire rilasciando il bus
              _delay_us (10);
              }
       }
```

```
/*******************************************************************************
Funzione di scrittura di un byte su bus 1-wire.
Scrive il byte passato come argomento
*******************************************************************************/
void one_wire_write_byte(unsigned short int data){
	int loop;	//ciclo di scrittura dei singoli bit nel file, LSb-first
	for (loop=0;loop<8;loop++) {
		one_wire_write_bit(data & 0x01);
		data >>= 1;//shifta a destra il byte per la lettura del bit successivo
		}
	}

/*******************************************************************************
Funzione di LETTURA di un dato sul dispositivo DS2751
attraverso il bus di comunicazione  1-wire.
Restitisce il valore giustificato a SX con bit di segno
*******************************************************************************/
int DS2751_read (char address){
	unsigned int Read_H, Read_L;
	int Read;
	one_wire_reset();
	one_wire_write_byte(SN_ADDRESS);//Selezione dell'unico disp. sul bus
	one_wire_write_byte(R_COMMAND); //Comando di lettura
	one_wire_write_byte(address);   //Indirizzo locazione da leggere
	Read_H = one_wire_read_byte();    //lettura MSB
	Read_L = one_wire_read_byte();    //lettura LSB

	//Composizione del risultato
	Read = Read_H & 0x00FF;
	Read = (Read << 8) & 0xFF00;
	Read |= (Read_L & 0x00FF);
	return Read;
	}

void DS2751_write (char address,char data) {
	one_wire_reset();
	one_wire_write_byte(SN_ADDRESS);//Selezione dell'unico disp. sul bus
	one_wire_write_byte(W_COMMAND);   //Comando di lettura
	one_wire_write_byte(address);     //Indirizzo locazione da scrivere
	one_wire_write_byte(data);        //Dato da scrivere
	}
```

spi_via_usi_driver.h

```
//functions declaration
void spiX_initmaster();
void spiX_transmission();
unsigned short spiX_read(volatile unsigned char address);
void spiX_write(volatile unsigned char address, unsigned char value);
void USART_Init();
void USART_Transmit( unsigned short data );
```

spi_via_usi_driver_.c

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include "spi_via_usi_driver.h"
#include "delay.h"

#define NOT_CS PB0

void spiX_transmission() {
	USISR |= (1<<USIOIF); //clearing counter interrupt flag
	do { //sotware toggling 3-wire mode
		USICR = (1<<USIWM0) | (1<<USICS1) | (1<<USICLK) | (1<<USITC);
		asm("nop");asm("nop"); //clock too fast...
		} while ((USISR & (1<<USIOIF)) == 0); //stop when 16 cycles are expired
	};

void spiX_initmaster() {
	DDRB |=  ((1<<PB6) | (1<<PB7));// Outputs.
	DDRB &= ~((1<<PB5) | (1<<PB1));// Inputs.
	PORTB |= (1<<PB5)|(1<<PB7);    // Pull-ups.
	}

void spiX_write(unsigned char address, volatile unsigned char value) {
	//setting write address
	address |= 0x80; //edit address: WRITE mode
	USIDR = address;// Put data address in USI data register.
	spiX_transmission();
	USIDR = value;// Put data in USI data register.
```

```
	spiX_transmission();
	}

unsigned short spiX_read(unsigned char address) {
	volatile unsigned short DataRead_H, DataRead_L, DataRead;
	USIDR = address;// send address in READ mode
	spiX_transmission();
	USIDR = 0x00; // send dummy byte
	spiX_transmission();
	_delay_us(100); //wait between subsequent transmission
	USIDR = address;// send address in READ mode (dummy)
	spiX_transmission();
	DataRead_H = USIDR; //read MSB
	USIDR = 0x00;
	spiX_transmission();
	DataRead_L = USIDR; //read LSB
	_delay_us(100); //wait between subsequent transmission
	//Assembling 16 bit data
	DataRead = DataRead_H & 0x00FF;
	DataRead = (DataRead << 8) & 0xFF00;
	DataRead |= DataRead_L & 0x00FF;
	return DataRead;
	}
```

uart_driver.c

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include "spi_via_usi_driver.h"

void USART_Init() {
	UBRRH = 0;    //Set baud rate to 115200 with no internal prescaler and fosc=8Mhz */
	UBRRL = 0x8; //error = -3.5%
	UCSRB = (1<<RXEN)|(1<<TXEN); //Enable TX & RX
	UCSRA = (1<<U2X); //Enable Double Speed
	UCSRC = (3<<UCSZ0); //Set frame format: 8data, no parity, 1 stop bit
	}


void USART_Transmit(volatile unsigned short data ) {
	volatile unsigned char data_L, data_H;
	data_L = data & 0x00FF; //De-assembling 16 bit data in 2 byte
	data_H = (data & 0xFF00) >> 8;
	while ( !( UCSRA & (1<<UDRE)) ); //Wait for empty transmit buffer
	UDR = data_H; //Put data into buffer: sends the data (MSB)
	while ( !( UCSRA & (1<<UDRE)) ); //Wait for empty transmit buffer
	UDR = data_L; //Put data into buffer: sends the data (LSB)
	}
```

Appendice C. Firmware misuratore di livello

pinout.h

```
/***USART PORT***/
#define PC 0
#define SENSOR 1
#define LED_PORT PORTC

/***LINEE***/
#define GaugeData 7
#define GaugeDataPin PIND
#define GaugeDataPort PORTD
#define GaugeDataDir DDRD
#define BT_Connected !(PINA&(1<<1))
#define PowerSupplyOn !(PINA&(1<<6))
#define ChargerFault  !(PINA&(1<<4))
#define Button !(PINA&(1<<7))
#define Sensor_powered !(PIND&(1<<5))

/***COMANDI***/
#define V24_OFF PORTD |= (1<<5)
#define V24_ON PORTD &= ~(1<<5)
#define SensorRead PORTD &= ~(1<<4)
#define SensorWrite PORTD |= (1<<4)
#define BT_ON PORTA &= ~(1<<0)
#define BT_OFF PORTA |= (1<<0)
#define CurrPullUpOFF PORTD &= ~(1<<6)
#define HiCurr DDRD |= (1<<6) //R in parallelo, connessa a massa, PIN OUTPUT
#define LoCurr DDRD &= ~(1<<6)//R disconnessa, PIN Hi-Z
#define Charger_OFF PORTA &= ~(1<<2)
#define Charger_ON  PORTA |= (1<<2)
#define C_int_enable PCMSK0|=(1<<PCINT6);
#define C_int_disable PCMSK0&=~(1<<PCINT6);
#define BLED_OFF PORTC |= (1<<4) //D2
#define BLED_ON PORTC &= ~(1<<4) //D2
#define GLED_ON PORTC |= (1<<5) //D3
#define GLED_OFF PORTC &= ~(1<<5) //D3
#define RLED_ON PORTC |= (1<<6) //D4
#define RLED_OFF PORTC &= ~(1<<6) //D4
```

main.h

```
//------------------------COUNTER0 compare match TOP VALUE definition-----------------------------------------
#define F_TICK 100UL // Tick freq=100Hz (Tick period=10ms)
#define PRESC 1024UL// using Fckio/1024 from PRESCALER
#define TIME_TOP_VALUE ((F_CPU/(PRESC * F_TICK ))-1) //OCR0A Compare Register Value to obtain Tick period=10ms
#define AUTO_OFF_TIME 18000 //0  ms
#define POSTSCALER_TOP 99
//------------------------------------------------------------------------------------------------------------

//------------------VARIABLE DEFINITIONS-----------------------------------------------------
uint16_t time;
volatile uint8_t tick_flag;
uint32_t press_count;
uint8_t push_done;
uint8_t postscaler;
//-------------------------------------------------------------------------------------------

enum com_status {IDLE, RX_PC, TX_SENSOR, RX_SENSOR, TX_PC} com_status_var ;
enum link_status {POWERDOWN, WAITING, CONNECTED, MEASURING} link_status_var ;
enum batt_status {OFF, CHARGING, FULL_CHARGE, BATT_ERROR} batt_status_var ;

int16_t gauge_data,charge,old_charge,voltage, current, temperature;
uint8_t interrupt_flag;
```

main.c

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <string.h>
#include "delay.h"
#include "commands.h"
#include "pinout.h"
#include "one_wire_driver.h"
#include "usart.h"
#include "main.h"
```

```
#include "led.h"
#include "i2c.h" //si faccia riferimento alla libreria AVR-LIB

//-----------------INTERRUPT ROUTINES-----------------------------------------------------------

//CTC mode TIMER0 interrupt service routine

ISR(TIMER0_COMPA_vect) {
	interrupt_flag=1;
	time++; //increasing waiting Bluetooth connection timer
	tick_flag=1; //set the while cycle syncronization variable
	if (com_status_var==RX_PC) timePC++;
	if (com_status_var==RX_SENSOR) timeSENSOR++;
	}

//Pushbutton interrupt service routine
ISR(PCINT0_vect) {
	interrupt_flag=1;
	if (Button) {
		time=0;
		press_count=0;
		_delay_us (1400);//wait 1->0 rebounds time  T1 = 800us
		while (Button) {
			press_count++; //long press detection
			if (press_count>400000) {
				link_status_var=POWERDOWN;
				Led_Routine_Set(RED,0,0,0);
				Led_Routine_Set(BLUE,0,0,0);
				Led_Routine_Set(GREEN,0,0,0);
				BLED_OFF;
				GLED_OFF;
				RLED_OFF;
				V24_OFF;
				BT_OFF;
				}
			}
		_delay_us (1400);//wait 0->1 rebounds time T2 = 400us
		push_done=1;
		}
	}

//Data received from PC
ISR(USART0_RX_vect) {
	interrupt_flag=1;
	if (bufferPCpointer<BufferSize-1) {
		bufferPC[bufferPCpointer]=UDR0;
		if (bufferPC[bufferPCpointer]=='\r') PC_End=1;
		bufferPCpointer++;
		}
	else bufferPCfull=1;
	}

//Data received from Sensor
ISR(USART1_RX_vect) {
	interrupt_flag=1;
	if (bufferSENSORpointer<BufferSize-1) {
		bufferSENSOR[bufferSENSORpointer]=UDR1;
		if (bufferSENSOR[bufferSENSORpointer]=='\r') Sensor_End=1;
		bufferSENSORpointer++;
		}
	else bufferSENSORfull=1;
	}

//---MAIN---
int main(void) {
	//TIMER SETTINGS
	TCCR0A  = (2<<WGM00); //CTC Mode
	TCCR0B =(5<<CS00);    //Prescaler 1024
	OCR0A = TIME_TOP_VALUE;// to obtain Tick period=10ms
	TIMSK0 = (1<<OCIE0A);  // Timer interrupts enabled

	//PORTS SETTINGS
	DDRA = 0b00001101;
	DDRB = 0;
	DDRC = 0b11111000;
	DDRD = 0b01111010;
	PORTC = 0;
	PORTA |= (1<<6)|(1<<4); //ACPR e FAULT pullup on

	//LEDs init.
	Led_Routine_Set(RED,0,0,0);
	Led_Routine_Set(BLUE,0,0,0);
```

```
Led_Routine_Set(GREEN,0,0,0);
BLED_OFF;
GLED_OFF;
RLED_OFF;

//USART init.
USART_Init();
bufferPCpointer=0;
bufferSENSORpointer=0;

//I2C init.
i2cInit();
i2cSetBitrate(300);
i2cMasterSendNI(0b10010010, 1, 0); //set reading addresses
i2cMasterSendNI(0b10010000, 1, 0);

//Pushbutton interrupt enable
PCICR=(1<<PCIE0);
PCMSK0=(1<<PCINT7);
set_sleep_mode(SLEEP_MODE_PWR_DOWN);
sei(); //Global interrupt enable

//Status init.
link_status_var = POWERDOWN;
batt_status_var = OFF;
com_status_var = IDLE;
Reset_buffer(PC);
Reset_buffer(SENSOR);
V24_OFF;
BT_OFF;
CurrPullUpOFF;
HiCurr;
Charger_ON;


// Infinite LOOP
while(1) {
       while (!tick_flag); //wait on tick flag
       tick_flag=0;  //reset flag

       Led_Routine();//LED updating

       //---COMMUNICATION STATE MACHINE---
       switch (com_status_var) {

              case IDLE:
              if (bufferPCpointer!=0) com_status_var=RX_PC;
              break;

              case RX_PC:
              if ((timePC>=USART_timeout)) {
                     SendPCerror(1);
                     com_status_var=IDLE;//reset
                     Reset_buffer(PC);
                     }
              if (PC_End){
                     if (bufferPCfull) {
                            SendPCerror(2);
                            com_status_var=IDLE;
                            Reset_buffer(PC);
                            }
                     else com_status_var=EXE_command(bufferPC);
                     }
                     break;

              case TX_SENSOR:
                     USART_send_buffer(SENSOR,bufferPC);
                     com_status_var=RX_SENSOR;
                     Reset_buffer(SENSOR);
                     break;

              case RX_SENSOR:
                     if ((timeSENSOR>=USART_timeout)) {
                            SendPCerror(3);
                            com_status_var=IDLE; //reset
                            Reset_buffer(PC);
                            }
                     if (Sensor_End) {
                            if (bufferSENSORfull) {
                                   SendPCerror(4);
                                   com_status_var=IDLE;
                                   Reset_buffer(PC);
```

```
                        }
                    else com_status_var=TX_PC;
                    }
                break;

        case TX_PC:
                USART_send_buffer(PC,bufferSENSOR);
                com_status_var=IDLE;
                Reset_buffer(PC);
                break;
        }

//---LINK STATE MACHINE---
switch (link_status_var) {

case POWERDOWN:
        while (Button); //wait for button release
        _delay_ms (1);//debounce
        Led_Routine_Set(RED,0,0,0);
        Led_Routine_Set(BLUE,0,0,0);
        Led_Routine_Set(GREEN,0,0,0);
        GLED_OFF;
        RLED_OFF;
        BLED_OFF;
        Charger_ON;
        V24_OFF;
        BT_OFF;
        C_int_enable; //enable charger interrupt
        sleep_mode();
        //...Goodnight...

        //...Goodmorning...
        C_int_disable;//disable charger interrupt
        time=0;
        BT_ON;
        link_status_var=WAITING;
        BLED_ON;
        RLED_OFF;
        charge=4000;old_charge=4000;
        while (Button); //wait for button release
        _delay_ms(1);//debounce
        push_done=0;
        break;

case WAITING:
        if (BT_Connected) {
                Led_Routine_Set(BLUE,LED_PERIOD_BT,LED_ONTIME_BT,0);
                link_status_var=CONNECTED;
                bufferPCpointer=0;
                push_done=0;
                }
        if ((time>AUTO_OFF_TIME)&&(!PowerSupplyOn)) link_status_var=POWERDOWN;
        break;

case CONNECTED:
        if (Sensor_powered){
                link_status_var=MEASURING;
                USART_send_buffer(PC,(uint8_t*) "24V ON\r");
                Led_Routine_Set(RED,LED_PERIOD_UF,LED_ONTIME_UF,0);
                Led_Routine_Set(GREEN,LED_PERIOD_UF,LED_ONTIME_UF,LED_ONTIME_UF);
                bufferPCpointer=0;
        }
        if (!BT_Connected) {
                link_status_var=WAITING;
                time=0;
                BLED_ON;
                Led_Routine_Set(BLUE,0,0,0);
                }
        break;

case MEASURING:
        batt_status_var=OFF;
        if (!Sensor_powered) {
                link_status_var=CONNECTED;
                Led_Routine_Set(RED,0,0,0);
                Led_Routine_Set(GREEN,0,0,0);
                RLED_OFF;
                GLED_OFF;
                USART_send_buffer(PC,(uint8_t*) "24V OFF\r");
                charge=4000;old_charge=4000;
                }
        if (!BT_Connected){
```

```
                link_status_var=WAITING; time=0;
                V24_OFF;
                BLED_ON;
                RLED_OFF;
                GLED_OFF;
                Led_Routine_Set(BLUE,0,0,0);
                Led_Routine_Set(GREEN,0,0,0);
                Led_Routine_Set(RED,0,0,0);
                charge=4000;old_charge=4000;
                }
        break;
        }
//si utilizza interrupt_flag per scartare i dati ONE-WIRE qualora la lettura venga interrotta da
                                                                        un interrupt
if (postscaler==POSTSCALER_TOP/4) {
        interrupt_flag=0;
        gauge_data=DS2751_read (I_REG);
        if (!interrupt_flag) current=gauge_data;
        }
if (postscaler==POSTSCALER_TOP*2/4) {
        interrupt_flag=0;
        gauge_data=DS2751_read (V_REG);
        if (!interrupt_flag) voltage=gauge_data;
        }
if (postscaler==POSTSCALER_TOP*3/4) {
        interrupt_flag=0;
        gauge_data=DS2751_read (T_REG);
        if (!interrupt_flag) temperature=gauge_data;
        }
if (postscaler==POSTSCALER_TOP) {
        postscaler=0;
        interrupt_flag=0;
        gauge_data=DS2751_read (C_REG);
        if (!interrupt_flag) {old_charge=charge; charge=gauge_data;}

        switch (batt_status_var) {

                case OFF:
                        if (!Sensor_powered) {//if powered, non ci si muove! //LSB 312.5 uAh
                                if (((charge<2*MIN_C)&&(old_charge>=2*MIN_C)) ||
                                                        ((charge>MIN_C)&&(old_charge<=MIN_C)))
                                        Led_Routine_Set(RED,LED_PERIOD_SLOW,LED_ONTIME_SLOW,0);
                                if ((charge<MIN_C)&&(old_charge>=MIN_C))
                                        Led_Routine_Set(RED,LED_PERIOD_FAST,LED_ONTIME_FAST,0);
                                if ((charge>2*MIN_C)&&(old_charge<=2*MIN_C)) {
                                        Led_Routine_Set(RED,0,0,0);
                                        RLED_OFF;
                                        }
                                if ((PowerSupplyOn)) {
                                        Charger_ON;
                                        batt_status_var=CHARGING;
                                        Led_Routine_Set(GREEN,LED_PERIOD_SLOW,LED_ONTIME_SLOW,0);
                                        Led_Routine_Set(RED,0,0,0);
                                        RLED_OFF;
                                        }
                                }
                        break;

                case CHARGING:
                        time=0;
                        if (!PowerSupplyOn) {
                                Charger_OFF;
                                batt_status_var=OFF;
                                Led_Routine_Set(GREEN,0,0,0);
                                Led_Routine_Set(RED,0,0,0);
                                RLED_OFF;
                                GLED_OFF;
                                charge=4000;old_charge=4000;
                                }
                        else {
                                if (ChargingRead()==2) {
                                        batt_status_var=FULL_CHARGE;
                                        Led_Routine_Set(GREEN,0,0,0);
                                        Led_Routine_Set(RED,0,0,0);
                                        RLED_OFF;
                                        GLED_ON;
                                        //USART_send_buffer(PC, (uint8_t*)"Carico\r");
                                        }

                                if ((ChargerFault)&&(DS2751_read (C_REG)>0)) {
                                        Charger_OFF;
                                        batt_status_var=BATT_ERROR;
```

```
                                        Led_Routine_Set(GREEN,0,0,0);
                                        Led_Routine_Set(RED,0,0,0);
                                        RLED_ON;
                                        GLED_OFF;
                                        }
                                }
                        break;

                case FULL_CHARGE:
                        time=0;
                        if (ChargerFault) {
                                Charger_OFF;
                                batt_status_var=BATT_ERROR;
                                Led_Routine_Set(GREEN,0,0,0);
                                Led_Routine_Set(RED,0,0,0);
                                RLED_ON;
                                GLED_OFF;
                                }
                        if (!PowerSupplyOn) {
                                Charger_OFF;
                                batt_status_var=OFF;
                                Led_Routine_Set(GREEN,0,0,0);
                                Led_Routine_Set(RED,0,0,0);
                                RLED_OFF;
                                GLED_OFF;
                                charge=4000;old_charge=4000;
                                }
                        break;

                case BATT_ERROR:
                        time=0;
                        Charger_ON;
                        _delay_ms(3);
                        if (!PowerSupplyOn) {
                                batt_status_var=OFF;
                                Led_Routine_Set(GREEN,0,0,0);
                                Led_Routine_Set(RED,0,0,0);
                                RLED_OFF;
                                GLED_OFF;
                                charge=4000;old_charge=4000;
                                }
                        Charger_OFF;

                break;
                }
                }
        else postscaler++;
        }
    }
```

commands.h

```
#define FULL_C 0x29 //312,5uAh/LSB
#define HALF_C 0x0A
#define QUARTER_C 0x05
#define MIN_C 1320

extern uint8 t V24 supply;
enum com_status EXE_command(uint8_t *buffer);
uint8_t ChargingRead (void);
void ResetCharger (void);
```

commands.c

```
#include <string.h>
#include <avr/io.h>
#include "usart.h"
#include "commands.h"
#include "pinout.h"
#include "main.h"
#include "one_wire_driver.h"
#include "i2c.h"
#include "delay.h"

const char version[]="01";

//esegue i comandi e carica i messaggi da inviare sul buffer coretto
//ritorna lo stato in cui proseguire
enum com_status EXE_command (uint8_t* buffer) {
    if (buffer[0]=='@') {//Se il comando è destinato al sensore verifica che sia acceso
        if (!Sensor_powered) {
            SendPCerror(5);
```

```
                Reset_buffer(PC);
                return IDLE;
                }
        else return TX_SENSOR;
        }
    else{  //Gli altri comandi sono gestiti direttamente dal uC e la risposta è caricata nel buffer

        if (strncmp((char*)bufferPC,"V\r",2)==0)  { //lettura tensione
                bufferSENSOR[0]='0';
                bufferSENSOR[1]='V';
                bufferSENSOR[2]=voltage>>8;
                bufferSENSOR[3]=voltage;
                bufferSENSOR[4]='\r';
                bufferSENSOR[5]='\0';
                }
        else if (strncmp((char*)bufferPC,"I\r",2)==0){ //lettura corrente
                bufferSENSOR[0]='0';
                bufferSENSOR[1]='I';
                bufferSENSOR[2]=current>>8;
                bufferSENSOR[3]=current;
                bufferSENSOR[4]='\r';
                bufferSENSOR[5]='\0';
                }
        else if (strncmp((char*)bufferPC,"C\r",2)==0){ //lettura carica
                bufferSENSOR[0]='0';
                bufferSENSOR[1]='C';
                bufferSENSOR[2]=charge>>8;
                bufferSENSOR[3]=charge;
                bufferSENSOR[4]='\r';
                bufferSENSOR[5]='\0';
                }
        else if (strncmp((char*)bufferPC,"T\r",2)==0){ //lettura temperatura (gauge)
                bufferSENSOR[0]='0';
                bufferSENSOR[1]='T';
                bufferSENSOR[2]=temperature>>8;
                bufferSENSOR[3]=temperature;
                bufferSENSOR[4]='\r';
                bufferSENSOR[5]='\0';
                }
        else if (strncmp((char*)bufferPC,"K\r",2)==0){ //azzeramento carica
                DS2751 write(C REG,0x00);
                DS2751 write(C REG+1,0x00);
                strcpy ((char*) bufferSENSOR, "!\r");
                }
        else if (strncmp((char*)bufferPC,"F\r",2)==0){ //scrittura carica massima
                DS2751_write(C_REG,FULL_C);
                DS2751_write(C_REG+1,0x40);
                strcpy ((char*) bufferSENSOR, "!\r");
                }
        else if (strncmp((char*)bufferPC,"D\r",2)==0){ //scrittura metà carica
                DS2751_write(C_REG,HALF_C);
                DS2751_write(C_REG+1,0x20);
                strcpy ((char*) bufferSENSOR, "!\r");
                }
        else if (strncmp((char*)bufferPC,"S\r",2)==0){ //scrittura 1/4 di carica
                DS2751 write(C REG,QUARTER C);
                DS2751_write(C_REG+1,0x20);
                strcpy ((char*) bufferSENSOR, "!\r");
                }
        else if ((strncmp((char*)bufferPC,"V24ON\r",6)==0)){//accensione sensore
                //il sensore viene acceso solo con dispositivo non in carica
                //e batteria sufficientemente carica
                if ((batt_status_var==OFF) && (charge>=MIN_C))  {
                        V24_ON;
                        strcpy ((char*) bufferSENSOR, "!\r");
                        }
                else strcpy ((char*) bufferSENSOR, "E6\r");
                }
        else if (strncmp((char*)bufferPC,"V24OFF\r",7)==0) {//spegnimento sensore
                V24_OFF;
                strcpy ((char*) bufferSENSOR, "!\r");
                }
        else if (strncmp((char*)bufferPC,"H\r",2)==0){ //ricarica con 500 mA
                HiCurr;
                strcpy ((char*) bufferSENSOR, "!\r");
                }
        else if (strncmp((char*)bufferPC,"L\r",2)==0){ //ricarica con 100 mA
                LoCurr;
                strcpy ((char*) bufferSENSOR, "!\r");
                }
        else if (strncmp((char*)bufferPC,"C ON\r",5)==0)   {//accensione caricabatteria
                Charger ON;
```

```
                    strcpy ((char*) bufferSENSOR, "!\r");
                    }
             else if (strncmp((char*)bufferPC,"C OFF\r",6)==0)  {//spegnimento caricabatteria
                    Charger_OFF;
                    strcpy ((char*) bufferSENSOR, "!\r");
                    }
             else if (strncmp((char*)bufferPC,"TA\r",3)==0){//lettura temperatura ambiente
                    uint8_t app_bytes[2]="\0\0";
                    i2cMasterReceiveNI(0b10010010, 2, (void*)app bytes);
                    strcpy ((char*) bufferSENSOR, "0T  \r");
                    bufferSENSOR[2]=app_bytes[0];
                    bufferSENSOR[3]=app_bytes[1];
                    }
             else if (strncmp((char*)bufferPC,"TB\r",3)==0){//lettura temperatura batteria
                    strcpy ((char*) bufferSENSOR, "0T  \r");
                    i2cMasterReceiveNI(0b10010000, 2, (void*)(&(bufferSENSOR[2])));
                    }
             else if (strncmp((char*)bufferPC,"VER\r",2)==0){ //versione firmware
                    strcpy ((char*) bufferSENSOR, "V__\r");
                    bufferSENSOR[1]=version[0];
                    bufferSENSOR[2]=version[1];
                    }
             else strcpy ((char*) bufferSENSOR, "UNKNOWN\r"); //Comando sconosciuto
             return TX_PC;
             }
       }

//Funzione di decodifica del pin /CHRG del caricabatteria (vedi datasheet)
uint8 t ChargingRead (void) { //valore di ritorno: 2=fine carica; 1=10% della corrente impostata; 0=in carica.
       DDRA |= (1<<3); PORTA |= (1<<3); //PA3=3.3V
       if (!(PINA & (1<<5))) return 0;
       else{
             DDRA &= ~(1<<3); PORTA &= ~(1<<3);//PA3=HiZ
             _delay_us(100);
             if (!(PINA&(1<<5))) return 1;
             else return 2;
             }
       }

void ResetCharger (void) {
       Charger OFF;
        delay ms(1);
       Charger ON;
       }
```

led.h

```
enum color {RED, GREEN, BLUE};


#define LED_FIXED 16
#define LED_PERIOD_BT 100
#define LED_ONTIME_BT 10
#define LED_PERIOD_SLOW 150
#define LED_ONTIME_SLOW 75
#define LED_PERIOD_FAST 50
#define LED_ONTIME_FAST 25
#define LED_ONTIME_UF 10
#define LED_PERIOD_UF 20
#define LED_OFF 0
#define LED_ON 2

void Led_Routine_Set(enum color led_color, uint8_t period_time, uint8_t on_time, uint8_t offset);
void Led_Routine();
```

led.c

```
#include <avr/io.h>
#include "main.h"
#include "led.h"
#include "pinout.h"

uint8_t red_period, red_ontime, green_period, green_ontime,
blue_period, blue_ontime, red_counter, green_counter, blue_counter;

void Led_Routine_Set (enum color led_color, uint8_t period,uint8_t ontime, uint8_t offset) {
       switch (led_color) {
             case RED: red_period=period; red_ontime=ontime; red_counter=offset; break;
             case GREEN: green_period=period; green_ontime=ontime; green_counter=offset; break;
             case BLUE: blue_period=period; blue_ontime=ontime; blue_counter=offset; break;
             }
       }
```

```
void Led_Routine () {
       red_counter++;
       green_counter++;
       blue_counter++;

       if ((red_counter)>=(red_period)) red_counter=0;
       else if (red_counter>=red_ontime) RLED_OFF; else RLED_ON;

       if ((green_counter)>=(green_period)) green_counter=0;
       else if (green_counter>=green_ontime) GLED_OFF; else GLED_ON;

       if ((blue_counter)>=(blue_period)) blue_counter=0;
       else if (blue_counter>=blue_ontime) BLED_OFF; else BLED_ON;
       }
```

one_wire_driver.h

Vedere Appendice B - Firmware inclinometro

one_wire_driver.c

Vedere Appendice B - Firmware inclinometro

i2c.h

Vedere AVR-lib [17]

i2c.c

Vedere AVR-lib [17]

usart.h

```
//-----------------USART CONSTANTS DEFINITIONS-----------------------------------------------
#define BufferSize 16
#define USART_timeout 100
//-------------------------------------------------------------------------------------------

uint8_t bufferPC [BufferSize],bufferSENSOR [BufferSize];
uint8 t bufferPCpointer,bufferSENSORpointer;
volatile uint8 t Sensor End,PC End,bufferPCfull,bufferSENSORfull;
uint16_t timePC, timeSENSOR;

void USART_Init() ;
uint8_t USART_send_buffer(uint8_t ID, uint8_t* buffer);
void Reset_buffer(uint8_t ID);
void SendPCerror(uint8 t);
```

usart.c

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include "delay.h"
#include "usart.h"
#include "pinout.h"

void USART_Init() {
       /*Setting baudrates UART0*/
       UCSR0A = (1<<U2X0); // Double speed ON
       UBRR0H=0;
       UBRR0L=3;
       /*Setting baudrates UART1*/
       UCSR1A = 0; // Double speed OFF
       UBRR1H=0;
       UBRR1L=3;
       /* Enable receivers and transmitters */
       UCSR0B = (1<<RXEN0)|(1<<TXEN0)|(1<<RXCIE0);
       UCSR1B = (1<<RXEN1)|(1<<TXEN1)|(1<<RXCIE1);
       /* Set frame format: 8data, 1stop bit */
       UCSR0C = (3<<UCSZ00);
       UCSR1C = (3<<UCSZ10);
       }

uint8_t USART_send_buffer(uint8_t ID, uint8_t* buffer) {
       uint8_t i=0;
       if (ID)  { //sensore
              SensorWrite;
              while ((buffer[i]!='\r')&&(i<BufferSize)) {
                     while ( !( UCSR1A & (1<<UDRE1)));
```

```
                    UCSR1A |= (1<<TXC1);
                    UDR1=buffer[i];
                    i++;
                    }
             if (buffer[i]=='\r') {
                    while ( !( UCSR1A & (1<<UDRE1)));
                    UCSR1A |= (1<<TXC1);
                    UDR1='\r';
                    }
             while ( !( UCSR1A & (1<<TXC1)));
             SensorRead;  //attesa ultima trasmissione, solo SENSORE!
             }
      else { //PC
             while ((buffer[i]!='\r')&&(i<BufferSize)) {
                    while ( !( UCSR0A & (1<<UDRE0)));
                    UDR0=buffer[i];
                    i++;
                    }
             if (buffer[i]=='\r') {
                    while ( !( UCSR0A & (1<<UDRE0)));
                    UDR0='\r';
                    }
             }
      return i;
      }

void Reset_buffer(uint8_t ID) {
      if (ID) {
             timeSENSOR=0;
             bufferSENSORpointer=0;
             bufferSENSORfull=0;
             Sensor_End=0;
             }
      else {
             timePC=0;
             bufferPCpointer=0;
             bufferPCfull=0;
             PC_End=0;
             }
      }

void SendPCerror(uint8_t ErrorNumber) {
      while ( !( UCSR0A & (1<<UDRE0))) ;
      UDR0='E';
      while ( !( UCSR0A & (1<<UDRE0))) ;
      UDR0=48+ErrorNumber;
      while ( !( UCSR0A & (1<<UDRE0))) ;
      UDR0='\r';
      }
```

Appendice D. Comunicazione col sensore magnetostrittivo

Estratto da:

Istruzioni per l'installazione ed uso dei trasduttori magnetostrittivi - Serie PC

Versione 1.0 rev.5 – DS Europe. [16]

14. IL PROTOCOLLO DI COMUNICAZIONE SERIALE

La serie PC può anche essere utilizzata mediante il collegamento remoto con un computer o con un PLC.

Si potranno:

- utilizzare le funzioni di misura del trasduttore in "remoto";
- eseguire la personalizzazione del trasduttore;
- configurare le uscite analogiche;

Mediante il semplice protocollo di comunicazione implementato, il trasduttore può far parte di una linea di trasmissione seriale multi-drop RS 485 composta da un computer (operante da *master*) ed uno o più sensori che opereranno da *slaves*. Durante la trasmissione seriale la funzione del protocollo è quella di garantire l'integrità dei dati trasmessi, siano essi comandi, misure, o risposte del trasduttore.

Tutte le operazioni sopra indicate vengono eseguite stabilendo una comunicazione tra il computer ed il trasduttore mediante l'utilizzo di una serie di comandi. Questi ultimi devono essere integrati in una stringa di comando ASCII con un formato ben preciso.

14.1. FORMATO DEL PROTOCOLLO

Il protocollo considera due formati diversi di messaggio da inviare al trasduttore della serie PC: uno per comandi generici ed uno per comandi di configurazione del trasduttore.

14.2. STRUTTURA DEL MESSAGGIO DI COMANDO

La struttura dei messaggi da computer al PCS/PCR, per i comandi, è la seguente: Ogni *comando* deve essere iniziato da un carattere ASCII di inizio messaggio, @ (0x40). Questo carattere "@" occuperà sempre la prima posizione nella costruzione del messaggio. Successivamente deve essere inserito l'identificativo (ID) del trasduttore a cui il comando è indirizzato. L'identificativo deve essere espresso da un carattere che può essere compreso tra "0" e "9", oppure da "A" a "Z" (maiuscole), oppure dal punto di domanda "?" qualora si desideri inviare un messaggio ad un trasduttore di cui non sia noto l'identificativo.

Il punto di domanda, con connessioni seriali multi-drop RS 485 (più PCS/PCR sulla stessa linea seriale), deve essere utilizzato avendo collegato sulla linea seriale soltanto un trasduttore alla volta, al fine di evitare che tutti quelli presenti rispondano al computer contemporaneamente, creando una situazione ingestibile.

La tipica applicazione del "?" consiste nel poter stabilire il collegamento con un trasduttore che non abbia identificativo, per potergliene quindi assegnare uno.

Dopo l'identificativo (sempre nella seconda posizione nella costruzione del messaggio) deve seguire un carattere equivalente al comando che si desidera impartire al PCS/PCR.

Questo carattere ASCII occuperà sempre la terza posizione nella costruzione del messaggio.

Per la descrizione dei comandi si rimanda al paragrafo 14.3.

Al comando inviato al trasduttore deve seguire, qualora sia presente, il suo argomento che può essere formato da uno o più caratteri ASCII a partire dalla quarta posizione del messaggio.

A chiusura del messaggio deve essere sempre posto “<cr>” (Carriage Return, 0x0D).

STRUTTURA MESSAGGIO DI COMANDO

@	ID	COMANDO	<cr>

14.3. I COMANDI AMMESSI

NOTA: Il trasduttore riconoscerà il carattere di comando inviato, indipendentemente dal fatto che la lettera ad esso equivalente sia maiuscola o minuscola.

Supponendo di collegarsi ad un trasduttore il cui identificavo sia “0”, saranno utilizzabili i seguenti comandi:

14.3.1. COMANDO “A”

Il comando “A” consente di modificare, od assegnare, l’identificativo (ID) del trasduttore.

I valori ammessi sono da “0” a “9” oppure da “A” a “Z” (maiuscole).

Esempio: @0A1<cr>

A conferma dell’esecuzione del comando inviato, cioè la sostituzione dell’ID da 0 a 1, il trasduttore risponderà con un punto esclamativo “!”.

Se non si ricorda l’identificativo del trasduttore è comunque possibile collegarsi con esso utilizzando il ?.

Esempio: @?A1<cr>

14.3.2. COMANDO “D”

Il comando “D” permette di impostare il byte di configurazione dei D\A converters, caratterizzandone quindi il funzionamento.

Il comando può essere riassunto dal formato “Dbbbbbbbb”, dove “D” è il comando vero e proprio e “bbbbbbbb” sono gli 8 bit che compongono il byte (ciascuna “b” può assumere solamente il valore “0” od il valore “1”). I bit sono numerati da destra a sinistra.

Il primo a destra è il numero 0 (sequenza 7-6-5-4-3-2-1-0).

BIT	D\A	Valore bit = 0	Valore bit = 1
0	1	CURSORE 0	CURSORE 1
1	1	SPOSTAMENTO	VELOCITÁ
2	1	DRITTO	INVERSO
3	1	SPENTO	ACCESO
4	2	CURSORE 0	CURSORE 1
5	2	SPOSTAMENTO	VELOCITÁ
6	2	DRITTO	INVERSO
7	2	SPENTO	ACCESO

Esempio : @ 0 D 10011000 <cr>

Nell’esempio il convertitore D/A 2 (1001) viene configurato come acceso, diritto, rappresenta la misura di spostamento per il cursore 1 mentre il D/A 1 (1000) vieneconfigurato come acceso, diritto, rappresentando la misura di spostamento per il cursore 0.

Le impostazioni sulle uscite analogiche hanno effetto solo se sono stati installati, a seguito di ordine del Cliente, i relativi convertitori D/A. Qualora i convertitori non fossero presenti l’impiego del comando “D” non ha effetto sulla funzionalità del trasduttore.

14.3.3. COMANDO “L”

Il comando "L" permettere di impostare il valore numerico dei riferimenti di ZERO e FONDO SCALA dei due cursori esprimendoli in millimetri (*od altre unità di misura*) in modo da ottenere per il dato digitale (RS485) una misura di spostamento, o di livello, espressa in unità meccaniche.
Il comando deve avere il formato riportato negli esempi, ovvero "L" seguito dal numero del cursore (0 od 1) a cui si riferisce l'impostazione che si intende eseguire, seguito infine da "L" se si desidera impostare il valore di ZERO od "H" se si desidera impostare il valore di FONDO SCALA.
L'argomento del comando deve essere un numero composto da 6 cifre, positivo (senza segno).
Nel caso in cui due cursori siano utilizzati sullo stesso stelo di misura per cursore "0" si intende quello più vicino all'elettronica (uscita cavo o connettore) mentre il cursore "1" indica quello più lontano (vedi figura 13).
NOTA IMPORTANTE : dopo aver inviato i nuovi parametri, perché questi abbiano effetto, è necessario spegnere e riaccendere il trasduttore in modo da permettere il ricalcolo dei coefficienti interni.
Esempio 1: @ 0 L0L 000000 <cr>
Si imposta il riferimento inferiore (ZERO) del *cursore 0* ad un valore di 0 (mm).
Esempio 2: @ 0 L0H 001300 <cr>
Si imposta il riferimento superiore (FONDO SCALA) del *cursore 0* ad un valore di 1300 (mm).
Esempio 3: @ 0 L1L 000100 <cr>
Si imposta il riferimento inferiore (ZERO) del *cursore 1* ad un valore di 100 (mm).
Esempio 4: @ 0 L1H 001500 <cr>
Si imposta il riferimento superiore (FONDO SCALA) del *cursore 1* ad un valore di 1500 (mm).

14.3.4. COMANDO "R"

Il comando "R" richiede, da computer remoto o PLC, il valore misurato per i cursori 0 ed 1.
L'argomento del comando "R" è un carattere ASCII che può assumere i seguenti valori:
"R0" qualora si richieda al trasduttore il valore del primo cursore (cursore 0, vicino all'elettronica).
Esempio 1: @ 0 R 0 <cr>
"R1" qualora si richieda al trasduttore il valore del secondo cursore (cursore 1).
Esempio 2: @ 0 R 1 <cr>

14.3.5. COMANDO "V"

Il comando "V" richiede la revisione del firmware installato a bordo del trasduttore.
Esempio: @ 0 V <cr>
Esempio di risposta: PC V.01.00 S/N xxxxxx
Alla ricezione del comando il trasduttore risponde inviando una stringa ASCII contenente la versione del firmware, la data di riferimento ed il suo numero di serie.

14.3.6. COMANDO "T"

Il comando "T" consente la taratura del trasduttore: memorizza quanto misurato dal trasduttore, quando il cursore è posizionato in corrispondenza del riferimento di ZERO o di FONDO SCALA, abbinando le misure ai riferimenti numerici impostati con i comandi "L".
Esempio 1 : @ 0 T0Z <cr>
Misura e memorizza la posizione del *cursore 0* nel punto di ZERO.
Esempio 2 : @ 0 T0F <cr>

Misura e memorizza la posizione del *cursore 0* nel punto di FONDO SCALA.
Esempio 3 : @ 0 T1Z <cr>
Misura e memorizza la posizione del *cursore 1* nel punto di ZERO.
Esempio 4 : @ 0 T1F <cr>
Misura e memorizza la posizione del *cursore 1* nel punto di FONDO SCALA.
NOTA IMPORTANTE : dopo aver memorizzato le nuove posizioni, perché queste abbiano effetto, è necessario spegnere e riaccendere il trasduttore in modo da permettere il ricalcolo dei coefficienti interni.
Per il corretto funzionamento del trasduttore è quindi necessario tarare il trasduttore in modo da rispettare gli abbinamenti qui riassunti:

	ZERO		**FONDOSCALA**	
	Imposta riferimento	**Memorizza misura**	**Imposta riferimento**	**Memorizza misura**
Cursore 0	@0L0Lxxxxxx	@0T0Z	@0L0Hxxxxxx	@0T0F
Cursore 1	@0L1Lxxxxxx	@0T1Z	@0L1Hxxxxxx	@0T1F

Nota 1: "xxxxxx" indica l'argomento a 5 cifre da inviare al trasduttore con il comando relativo.
Nota 2: Ogni comando inviato al trasduttore deve essere terminato dal <cr> (Carriage Return , 0x0D).

14.3.7. COMANDO "X"

Il comando "X" consente di leggere il contenuto della memoria di impostazione EEPROM.
In questa memoria vengono inseriti, in modo permanente, i parametri di configurazione del trasduttore. Questi sono modificati e memorizzati tramite i comandi sopra citati.
Il significato dei parametri e la loro locazione è indicato in tabella.

INDICE	**PARAMETRO**	**NOTE**	**COMANDO DI IMPOSTAZIONE**
0	Limite 0 Low	Riferimento di ZERO cursore 0	*@0L0Lxxxxxx*
1	Limite 0 High	Riferimento del FONDO SCALA cursore 0	*@0L0Hyyyyyy*
2	Taratura Minimo 0	Valore di conteggio interno Minimo 0	*@0T0Z*
3	Taratura Massimo 0	Valore di conteggio interno Massimo 0	*@0T0F*
4	Limite 1 Low	Riferimento di ZERO cursore 1	*@0L1Lzzzzzz*
5	Limite 1 High	Riferimento del FONDO SCALA cursore 1	*@0L1Hwwwww*
6	Taratura Minimo 1	Valore di conteggio interno Minimo 1	*@0T1Z*
7	Taratura Massimo 1	Valore di conteggio interno Massimo 1	*@0T1F*
8	Configurazione D/A	Bit di configurazione D/A converters	*@0Dbbbbbbbb*
9	Indirizzo seriale	ID seriale del trasduttore	*@0Ax*

Esempio: @ 0 X 0 <cr>
Con l'esempio viene letto il contenuto del parametro 0 (Riferimento di ZERO cursore 0)
Nota: "xxxxxx", "yyyyyy", "zzzzzz", "wwwwww" sono argomenti a 6 cifre dei comandi utilizzati per impostare i parametri di configurazione del trasduttore.

14.4. RISPOSTE DEL TRASDUTTORE

Il trasduttore risponderà sempre ad un comando ad esso inviato con le seguenti stringhe di risposta, in funzione del comando inviato ed a seconda del contesto.

RISPOSTA DAL PCS/PCR	SIGNIFICATO
! <cr>	Il trasduttore risponde ad un comando ad esso inviato con un punto esclamativo quando il comando viene accettato ed eseguito.
? <cr>	Il punto di domanda è la risposta del trasduttore ad un comando sconosciuto o qualora il formato utilizzato non sia corretto. Il comando che ha generato come risposta il punto di domanda non è stato eseguito.
yRxxxxxxx<cr> Es. 1: 0R0120500 Es. 2: 1R-203450	E' la risposta al comando di richiesta del valore della misura (Ry, vedi 13.3.4), dove "y" indica il numero del cursore (0 od 1) e "xxxxxxx" corrisponde al valore della misura richiesta, comprensiva di segno. Quanto misurato è espresso nelle unità meccaniche utilizzate per definire i parametri di configurazione (comando L). La misura è fornita senza punti decimali. Se, per esempio, sono stati definiti dei parametri di configurazione espressi in decimi di millimetro, il valore della misura di posizione verrà restituito in decimi di millimetro. Il programma che interroga il trasduttore, per conoscere la posizione dei cursori, dovrà effettuare la gestione del punto decimale coerentemente con le scelte effettuate in sede di configurazione.
yR9999999<cr>	E' la risposta al comando "Ry", di richiesta della misura di posizione per il cursore y, qualora il cursore in questione non venga rilevato dal trasduttore (il cursore è stato sfilato dallo stelo di misura di un trasduttore PCS) Nel caso venissero utilizzati due cursori sullo stesso stelo di misura, questa risposta può anche indicare una distanza tra i cursori inferiore a quella minima accettabile.
xXzzzzzzz<cr> Es.: 1X0001000	E' la risposta al comando "X", di richiesta della lettura di un parametro di configurazione, dove "x" indica il parametro richiesto e "zzzzzzz" corrisponde al valore del parametro (7 caratteri). Il valore del parametro, non ha spazi vuoti, e qualora questo avesse un numero di caratteri inferiore al formato definito, vengono utilizzati degli zeri di riempimento.